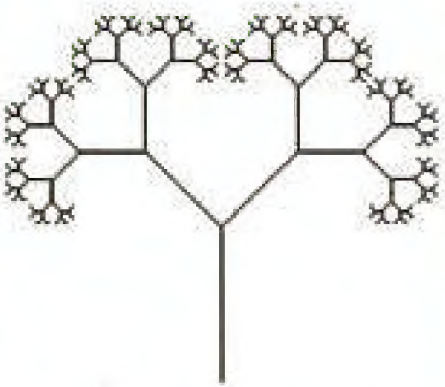
COMPUTER-PRAXIS MATHEMATIK

H. Göhner / B. Hafenbrak

Arbeitsbuch Q-BASIC

Programmbeispiele der Sekundarstufe I



● Rechnen ● Zeichnen ● Textbearbeitung
● mit Ausblicken auf Visual-BASIC

ÜMMLER

PDF-Konvertierung von Thomas Antoni – http://www.qbasic.de

COMPUTER-PRAXIS MATHEMATIK
Herausgegeben von StD Dietrich Pohlmann

H. Göhner / B. Hafenbrak

Arbeitsbuch Q-BASIC

Programmbeispiele der Sekundarstufe I:
● Rechnen ● Zeichnen ● Textbearbeitung
● mit Ausblicken auf Visual-BASIC

Zweite, durchgesehene und an
die neue Rechtschreibung angepasste Auflage.
Mit 327 Arbeitsaufträgen
und zahlreichen Lösungsbeispielen
sowie 74 Abbildungen.
Dümmlerbuch 4532

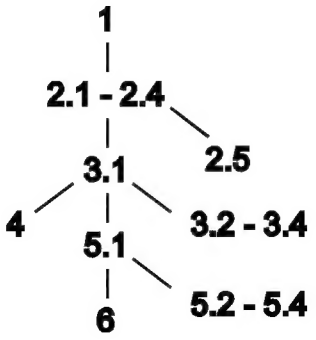
FERD. DÜMMLER'S VERLAG · BONN

Vorwort zur ersten und zweiten Auflage

Dieses Buch wendet sich an Schülerinnen und Schüler der Klassen 7 - 10 und ihre Lehrenden. Es will in den Umgang mit Algorithmen einführen und ihre Umsetzung in Programme lehren. Dies geschieht bevorzugt an Problemen der Mathematik, aber auch anhand von Spielen und im Umgang mit Texten. Die Programmiersprache Q-BASIC wurde dafür aus folgenden Gründen gewählt:

- Die Formulierung der Problemlösung in Q-BASIC lässt die Lösungsidee klar hervortreten und bleibt nahe bei einer umgangssprachlichen Formulierung.
- Die Symbolik stimmt mit der gewohnten Symbolik, etwa des Mathematikunterrichts, überein.
- Die Syntax ist durchsichtig, nicht rigide und hat eine auch für Schüler erkennbare und akzeptierbare Logik.
- Es ist möglich, ein Problem in Teilprobleme zu zerlegen und die Lösung der Teilprobleme zu einer Lösung des Gesamtproblems zusammenzusetzen.
- Der Schüler kann computernah arbeiten, was in dieser Altersstufe sehr wichtig ist. Auch Teillösungen können getestet werden.
- Die Eingabe und die Korrektur von Programmen ist sehr einfach.
- Frühe und recht präzise Fehlermeldungen helfen dem Benutzer.
- Die Sprache vermittelt einen Eindruck von der Denk- und Arbeitsweise in der Praxis und in der Wissenschaft. Sie erleichtert den Übergang zu professionellen Systemen.
- Q-BASIC wird mit MS-DOS bzw. Windows umsonst ausgeliefert, verursacht also keine zusätzlichen Kosten.

Das Buch ist in acht Kapiteln gegliedert. Das erste Kapitel führt in notwendige Grundkenntnisse und -fertigkeiten ein, die ständig benötigt werden. Die meisten Leser werden dieses Kapitel nur überfliegen, da die dort vermittelten Fertigkeiten heute oft bei anderen Computeranwendungen gelernt werden. Der eigentliche Einstieg in die Arbeit mit dem Computer geschieht in Kapitel 2 über das Rechnen. In der unten stehenden Abbildung ist angedeutet, welche Teile des Buches unbedingt durchgearbeitet werden sollten, und welche zur Vertiefung und Ergänzung dienen.



Kapitel 7 behandelt als weitere, anspruchsvolle Vertiefung die Rekursion, Kapitel 8 will dazu verführen, erste Schritte in Visual-BASIC zu tun, und dabei die Kenntnisse aus Q-BASIC anzuwenden.

Q-BASIC ist eine sehr vielseitige Sprache. Das Buch schöpft nur einen kleinen Teil seiner Möglichkeiten aus. Unser Ziel ist aber kein Grundkurs Informatik und kein Sprachenkurs Q-BASIC, vielmehr wollen wir Schüler der Sekundarstufe I und ihre Lehrer an algorithmische Problemlösungen heranführen.

Wir bedanken uns bei dem Herausgeber, Herrn StD D. Pohlmann, für zahlreiche Anregungen und Verbesserungsvorschläge und beim Verlag Ferd. Dümmler für die freundliche Betreuung. Frau Ingrid Sattler danken wir für die einfallsreichen Illustrationen und Herrn Karl Otto Stroppel für die Hilfe beim Korrekturlesen. Schließlich danken wir den Studierenden und den Lehrerinnen und Lehrern, die uns bei unseren Kursen wertvolle Rückmeldungen gegeben haben.

Wir freuen uns, dass nach recht kurzer Zeit eine Neuauflage erforderlich wurde. Dabei wurden nur einige Unstimmigkeiten und Druckfehler berichtigt und die neue Rechtschreibung berücksichtigt.

Heidelberg und Weingarten Hartmut Göhner Bernd Hafenbrak

Inhaltsverzeichnis

1 Einführung - Erste Programme
1.1 Die Tastatur 7
1.2 Der Bildschirm von Q-BASIC und der Editor 7
1.3 Der Direktmodus 9
1.4 Erste Q-BASIC-Programme 12

2 Rechnen mit Q-BASIC
2.1 Lineare Programme 16
2.2 Wiederholungen 18
2.3 Entscheidungen 25
2.4 Funktionen und Prozeduren 31
2.5 Ergänzungen 37

3 Koordinatengrafik
3.1 Koordinatensysteme 47
3.2 Funktionsgraphen 53
3.3 Muster der Teilbarkeit 54
3.4 Magic-Eye-Bilder 56

4 Igelgrafik
4.1 Igelbefehle im Direktmodus 59
4.2 Vielecke 60
4.3 Prozeduren 62
4.4 Kreisprozeduren 63
4.5 Projekt Buchstaben 66

5 Textbearbeitung
5.1 Textvariable und Operationen mit Textvariablen 68
5.2 Spiele 72
5.3 Geheimschriften 74
5.4 Textanalysen 78

6 Felder und Dateien
6.1 Felder 86
6.2 Dateien (Files) 94

7 Rekursion
7.1 Rekursive Funktionen 96
7.2 Rekursive Prozeduren 99

8 Ausblick auf Visual-BASIC
8.1 Die wichtigsten Objekte 103
8.2 Einige Projekte 108

Anhang
Lösungen und Programmbeispiele 113
Kurzbeschreibung von Q-BASIC 130

Literaturverzeichnis 138
Stichwortverzeichnis 139

1 Einführung - Erste Programme

1.1 Die Tastatur

Wir beziehen uns auf die heute gebräuchlichste Tastatur mit deutschen Beschriftungen, wie sie auf der gegenüberliegenden Seite skizziert ist.. Du findest zuerst den **Schreibmaschinenblock** (den sog. alphanumerischen Teil), der im Wesentlichen wie bei einer elektrischen Schreibmaschine angeordnet ist und den du benutzt, um Texte und Zahlen einzugeben. Daneben sind in diesem Block noch einige Sondertasten, insbesondere die Tasten STRG, ALT und ALTGR, die nur mit anderen Tasten zusammen gebraucht werden. Wir werden in Q-BASIC diese Tasten vor allem in dem Rahmen nutzen, wie sie von Windowsprogrammen allgemein belegt sind (z.B. zum Kopieren und Einfügen). Die UMSCHALTTASTE ⇑ (auch HOCHSTELLTASTE oder SHIFT-Taste) benutzt du wie bei der Schreibmaschine zum Abruf der Zweitbelegung der Tasten (z.B. der Großbuchstaben). Wichtig ist die EINGABE-Taste (oder RETURN-Taste), die große Taste am rechten Rand des Schreibmaschinenblocks, meist mit ↵ beschriftet. Sie wirkt einerseits wie die Wagenrücklauftaste der elektrischen Schreibmaschine (du beginnst damit eine neue Zeile), wird außerdem häufig zur Bestätigung von Kommandos gebraucht. Zuletzt noch die KORREKTURTASTE ← (auch BACKSPACE -Taste genannt) am oberen rechten Rand des Schreibmaschinenblocks. Du verwendest sie vor allem zur raschen Korrektur bei der Eingabe.

Oben über der Schreibmaschinentastatur etwas abgesetzt ist der sog. **Funktionstastenblock** mit den Tasten F1 bis F12, der ESC-Taste und ganz rechts der PAUSE/UNTBR-Taste, mit der ein laufendes Programm gestoppt werden kann. Wichtige und viel gebrauchte Kommandos sind auf bestimmte Funktionstasten gelegt (z.B. F1 um die Hilfe abzurufen, F5 um ein Programm zu starten).

Ganz rechts der sog. **Zahlenblock**, den wir allerdings *nicht* zur Eingabe von Zahlen verwenden. Wir ignorieren diesen Block vollständig und benutzen nur die Tasten zwischen Schreibmaschinenblock und Zahlenblock: Mit den Cursorsteuertasten bewegst du den Cursor in die angegebene Richtung, mit BILD↑ und BILD↓ blätterst du in deinem Text, mit ENTF löschst du das Zeichen unter dem Cursor, die Einfüge-Taste EINFG wechselt zwischen Überschreibe- und Einfügemodus und wird zusammen mit der SHIFT-Taste benutzt, um Text aus der sog. Zwischenablage einzufügen. Die Tasten POS1 und ENDE schließlich bringen den Cursor an den Anfang bzw. an das Ende der Zeile, in der er sich gerade befindet. Insgesamt haben die Tasten in Q-BASIC also dieselbe Funktion, wie in den meisten Textverarbeitungsprogrammen.

1.2 Der Bildschirm von Q-BASIC und der Editor

Falls du deinen Rechner noch mit DOS betreibst, so kannst du Q-BASIC von der DOS-Ebene aus mit **qbasic** aufrufen. In Windows 3.x findest du Q-BASIC im Verzeichnis DOS. Du kannst die Datei qbasic.exe mit Hilfe des Dateimanagers aufrufen. Eine andere Möglichkeit ist die feste Einrichtung des Programms mit Hilfe des Programm-Managers. In Windows 95 befindet sich Q-BASIC auf der Installations-CD im Verzeichnis other\oldmsdos. Du musst die beiden Dateien qbasic.exe und q.basic.hlp auf deine festplatte kopieren, z.B. in das Verzeichnis c:\windows\command. Du kannst dann beim Start-Menü unter „Ausführen“ den Befehl **qbasic** abschicken.

Q-BASIC begrüßt dich mit seinem Eingangsbildschirm und fordert dich auf, durch Drücken der EINGABE-Taste eine kurze Einführung zu lesen oder mit der ESC-Taste gleich mit der Arbeit zu beginnen.

Drücke also ESC. In Windows 95 musst du vielleicht noch ALT+RETURN drücken, um den vollen Bildschirm zu erhalten. Du siehst jetzt den Q-BASIC-Bildschirm vor dir:

- Oben die Menüzeile, gestaltet wie bei allen Windows-Programmen,
- darunter das eigentliche Arbeitsfeld, in welches du bald deine BASIC-Programme schreiben wirst. Es ist überschrieben mit **Unbenannt** , es ist ja auch noch leer (sobald du ein Programm geschrieben und abgespeichert hast oder wenn du ein Programm lädst, steht der Name des Programms an dieser Stelle).
- Das untere Viertel des Arbeitsfelds ist ein zweites Fenster, überschrieben mit **Direkt**. Hier kannst du Befehle von Q-BASIC im sog. Direktmodus erproben.
- Unten schließlich die Infozeile mit Informationen über die möglichen Aktionen.

```
 Datei  Bearbeiten  Ansicht  Ausführen Suchen  Debug  Optionen  Hilfe
------------------------------------ Unbenannt ----------------------------------------
 UMSCH+F1=Hilfe  F6=Fenster  F2=SUBs  F5=Ausf  F8=Schritt  | N 00001:001
```

1) Probiere die in der Infozeile aufgeführten Funktionstasten aus:
   UMSCHALTTASTE+F1 ruft die Hilfe auf.
   F6 wechselt zwischen den beiden Fenstern des Arbeitsfelds.
   F2 wechselt zu einem Dialogfenster, welches den Wechsel zu Unterprogrammen steuert. Da kein Programm und Unterprogramm geschrieben oder geladen wurden, findest du natürlich noch kein Angebot.
   F5 führt das im Arbeitsfenster stehende Programm aus und wechselt zum Ausgabefenster. Da im Arbeitsfenster kein Programm steht, ist das Ausgabefenster leer (du erkennst das Ausgabefenster an der schwarzen Hintergrundfarbe).

2) Schaue dir die Menüs der Menüzeile an.
   Über die Tastatur rufst du sie so auf:
   Drücke die Alt-Taste und dann den jetzt hervorgehobene Buchstaben des Menüs, das dich interessiert, also z.B. Alt+D um das Menü Datei zu aktivieren.
   Einfacher geht es mit der Maus:
   Klicke das Menüfeld an, das dich interessiert.
   Verlasse Q-BASIC (über *Datei - Beenden*)

Zusammengefasst:

Das Menü **Datei** entspricht weitgehend dem gleichnamigen Befehl in anderen Windows-Programmen. Du benötigst es zum Laden (Öffnen), Speichern oder Drucken von Programmen. Hier findest du auch **Beenden**, den Befehl um Q-BASIC zu verlassen.

Die übrigen Menüs: **Bearbeiten** und **Suchen** bieten nützliche Möglichkeiten, wie bei einer Textverarbeitung (Kopieren, Suchen, Ändern), **Ansicht** ermöglicht einen Wechsel zwischen den verschiedenen Fenstern von Q-BASIC, **Ausführen** und **Debug** sind für die Programmausführung nützlich, mit **Optionen** kannst du z.B. Bildschirmeinstellungen (etwa Hintergrundfarben u.a.) ändern.

Die wichtigsten Befehle sind sowohl über das Menü wie über Funktionstasten ausführbar.

Die **Hilfe** ist wirklich sehr hilfreich, im Moment aber eher verwirrend, die Vielzahl der Schlüsselwörter schreckt dich vielleicht ab. Du wirst nur einen kleinen Teil davon wirklich brauchen, um sinnvoll zu arbeiten. Während du an diesem Buch arbeitest, kannst du dir zu den besprochenen Schlüsselwörtern mit der **Hilfe** noch weitere Informationen geben lassen, indem du über den **Index** das Wort suchst. Dazu gibst du den ersten Buchstaben des gesuchten Wortes ein.

3) Rufe Q-BASIC wieder auf und übe auf dem Arbeitsfeld "Unbenannt" die Textverarbeitungsfunktionen von Q-BASIC. Schreibe (mit den Fehlern):
   ´Dis ist ein feler und dis sind noch mer feler
   ´Jetzt beginnt die zweite Zeile
   ´dies ist eine weitere Zeile
   Bewege den Cursor mit den "Pfeiltasten" in die erste Zeile, mit POS1 bzw. ENDE kommst du an den Anfang bzw. das Ende dieser Zeile, bringe den Cursor auf die fehlerhaften Textstellen und korrigiere den Text.
   Einfacher gehts mit der Maus: fragliche Stelle links anklicken, schon ist der Cursor dort.

4) Markiere den Textabschnitt "*Dies ist ein Fehler...*"
   - mit der Tastatur: Cursor an den Anfang und mit UMSCHALTASTE+CURSOR RECHTS den Text überfahren
   - mit der Maus: Anfang des zu markierenden Bereichs anklicken, danach bei gedrückter UMSCHALTTASTE das Ende des zu markierenden Bereichs anklicken. Oder: mit gedrückter linker Maustaste den Text überfahren.

   Mit UMSCHALTTASTE+ENTF löschst du den Text und kopierst ihn gleichzeitig in die Zwischenablage, mit UMSCHALTTASTE+EINFG kopierst du ihn wieder aus der Zwischenablage an die Cursorstelle. Mit ENTF allein löscht du den markierten Text unwiederbringlich.

5) Füge eine dritte Zeile ein: "*´Dies ist die Zeile Nummer drei*"
   Bringe den Cursor ans Ende von Zeile 2, mit der EINGABE-Taste schaffst du Platz für die gewünschte dritte Zeile.

1.3 Der Direktmodus

Rechnen

Der Computer soll für dich die Subtraktionsaufgabe 27-19 rechnen.

6) Wechsle ins Fenster **Direkt** (anklicken oder F6) und gib ein **PRINT** 27-19

Mit Drücken der EINGABE-Taste (RETURN-Taste) wird der Befehl sofort ausgeführt: Q-BASIC wechselt in das Ausgabefenster und zeigt dort das Ergebnis 8 an. Durch Drücken irgendeiner Taste kommst du wieder zum Eingabefenster zurück.

PRINT ist der erste Befehl der Programmiersprache Q-BASIC, den du kennen lernst. Der besseren Übersichtlichkeit wegen werden wir Q-BASIC-Befehle immer groß schreiben, du kannst sie aber ruhig mit Kleinbuchstaben eintippen. **PRINT** bewirkt hier, dass der nachfolgende Rechenausdruck ausgewertet und das Ergebnis auf den (Ausgabe-)Bildschirm geschrieben wird. (Nach **PRINT** muss übrigens ein Leerzeichen folgen!) Der Befehl wird aber erst ausgeführt, wenn du ihn mit der EINGABE-Taste "abgeschickt" hast. Die EINGABE-Taste bewirkt:

- Einen "Wagenrücklauf" (engl. "carriage return") und "Zeilenvorschub", d.h. der Cursor geht an den Anfang der nächsten Zeile.
- Im Direktmodus wird eine Befehlszeile "abgeschickt"

7) Gib ein und stelle das Ergebnis fest:

PRINT 30.4-19.75
PRINT 12+5*4
PRINT 120/48
PRINT 2^3
PRINT 3^2

Fehlermeldungen

Hast du einmal das Leerzeichen zwischen PRINT und dem nachfolgenden Rechenausdruck vergessen, stellt Q-BASIC einen Fehler fest. Du bringst den Cursor mit der Cursortaste auf die erste Ziffer und schaffst eine Leerstelle durch Tippen der Leertaste. (Q-BASIC ist standardmäßig im sog. Einfügemodus, durch Drücken der EINFG-Taste kannst du in den Überschreibemodus wechseln, den wir dir aber nicht empfehlen.)
Du hast dich vielleicht einmal bei der Eingabe des Wortes **PRINT** vertippt. Wieder entdeckt Q-BASIC einen Syntaxfehler. Du korrigierst deinen Fehler und drückst die EINGABE-Taste..
Hast du einmal statt des Dezimalpunktes ein Komma eingegeben, so kommt keine Fehlermeldung, aber das angezeigte Resultat stimmt nicht. Ursache: Die "Grammatik" des **PRINT** Befehls erlaubt ein Komma, es hat aber nicht die Bedeutung des Dezimalkommas, sondern das eines Trennzeichens, wie du bald lernen wirst.

8) Mache absichtlich die eben beschriebenen Fehler und beobachte die Reaktion des Computers.

9) Inzwischen ist der (Ausgabe-)Bildschirm schon ziemlich voll geschrieben. Du löschst ihn mit CLS (und EINGABE-Taste drücken). CLS ist die Abkürzung für 'clearscreeen'.

Rechenart	Rechenzeichen auf dem Computer
addieren	+
subtrahieren	-
multiplizieren	*
dividieren	/
potenzieren	^
Potenzrechnung vor Punktrechnung vor Strichrechnung.	
Kommazahlen mit Dezimalpunkt eingeben.	
Ziffer 0 und Buchstabe O nicht verwechseln.	

10) Berechne mit dem Computer und kontrolliere durch Kopfrechnung.
17-3∗5+28; 100∗10-5∗20; (20+80)∗12-4∗25; 12,5∗8/2,5

11) Der Mittelwert von 3,7 und 5,08 soll berechnet werden. Welcher Befehl ist richtig?
a) **PRINT** 3,7+5,08/2
b) **PRINT** (3,7+5,08)/2
c) **PRINT** (3.7+5.08)/2

Text, Trennzeichen

12) Tippe ein. Jede Eingabe ist durch Drücken der EINGABE-Taste abzuschließen. (Diesen Hinweis lassen wir in Zukunft weg.)
PRINT "Q-BASIC"
PRINT "Hallo, hier ist dein Computer"
PRINT "12+5"
PRINT 12+5

Was zwischen Anführungszeichen steht, wird als Text (Zeichenfolge, engl. string) aufgefasst und wird mit PRINT unverändert ausgegeben.

Du kannst die Ausgabe von Text und Zahlen mischen, allerdings musst du die einzelnen13) Gib die folgenden Befehle ein:
PRINT "Die Summe von 31 und 17 ist";31+17
PRINT "31+17=";31+17
PRINT "31∗17=";31∗17

14) a) Ersetze in Aufgabe 13 jedes Semikolon (;) durch ein Komma (,). Welcher Unterschied im Ausdruck ergibt sich?
b) Tippe ein:
PRINT 1;2;3;4;5
PRINT 1,2,3,4,5
PRINT "3+4 =";3+4
PRINT "3+4 =",3+4

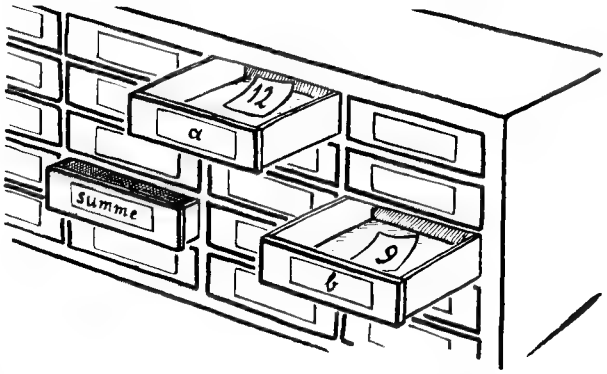
15) Sage die Anzeige des Computers voraus. Überprüfe dann.
a) **PRINT** 12,5∗40 b) **PRINT** 12.5∗40 c) **PRINT** 12;5∗40
d) **PRINT** 2,5∗2,5 e) **PRINT** 2.5∗2.5 f) **PRINT** 2;5∗2;5

1.4 Erste Q-BASIC-Programme

Bisher wurde jeder eingegebene Befehl vom Rechner unmittelbar nach der Eingabe ausgeführt ("Direktmodus"). Jeder Computer kann nun Befehle, die er erst später ausführen soll, speichern. Eine Folge solcher gespeicherter Befehle nennen wir ein "Programm". Ein Q-BASIC-Programm schreiben wir in das obere Fenster (im Moment noch mit "**Unbenannt**" überschrieben).

Beispiel:

```
' Erstes Programm
CLS
a=12
b=9
summe=a+b
PRINT summe
END
```



Ein Q-BASIC-Programm besteht aus einer Folge von Zeilen. Jede Zeile enthält einen Befehl an den Computer. Bei der Programmausführung werden die Befehle in der notierten Reihenfolge ausgeführt. Aus welchen Befehlen besteht unser Programm?

1. Zeile: Die Programmüberschrift. Der Rechner ignoriert alles, was auf das Zeichen ' (eine Abkürzung für **REM** von engl. 'remark') folgt, hier also die ganze Zeile.

2. Zeile: Der (Ausgabe-) Bildschirm wird gesäubert. (**CLS** von engl. 'clearscreen')

3. Zeile: Eine der vielen "Schubladen" (Speicher) des Computers erhält den Namen **a** und die Zahl 12 wird in diese Schublade abgelegt.

4. Zeile: Eine zweite "Schublade" erhält den Namen **b** und die Zahl 9 wird in diese Schublade abgelegt.

5. Zeile: Der Computer liest die Inhalte der Speicher **a** und **b**, addiert sie und bringt das Ergebnis in einen Speicher, der den Namen **summe** erhält.

6. Zeile: Der Inhalt des Speichers **summe** wird ausgegeben.

7. Zeile: Das Programm wird mit **END** beendet. Dieser Befehl kann auch weggelassen werden. In den folgenden Kapiteln lassen wir ihn aus Platzgründen weg.

16) Gib das Beispielprogramm ein. Schreibe alles in Kleinbuchstaben. Nach jeder Zeile drückst du die EINGABE-Taste, damit du an den Anfang der nächsten Zeile kommst. Sobald du in die neue Zeile wechselst, werden die Schlüsselwörter (hier CLS und PRINT) automatisch in Großbuchstaben umgewandelt. Zum Starten des Programms drücke die Taste F5 (oder wähle im Menü: **Ausführen Start).** Jetzt erscheint blitzschnell das Ergebnis "21" im Ausgabefenster.

17) Lass dir im Direktmodus den Inhalt der Speicher **a**, **b** und **summe** zeigen: Wechsle durch Anklicken oder mit F6 ins **Direkt**-Fenster. Lösche dessen Inhalt (Markieren und ENTF-Taste drücken). Gib ein: **PRINT a**
PRINT b
PRINT summe

18) Ergänze dein Programm, sodass auch Differenz, Produkt und Quotient der Zahlen in den Speichern a und b berechnet, in eigenen Speichern abgelegt und ausgegeben werden. Wechsle dazu wieder ins

Programmfenster und schreibe zusätzliche Programmzeilen. Starte das ergänzte Programm mit SHIFT F5.

Die Wertzuweisung

Wir wollen uns die Befehle in den Zeilen 3-5 unseres Beispielprogramms noch genauer ansehen. Das Zeichen = kommt in jeder dieser Zeilen vor. Es heißt *Wertzuweisung*. Der Befehl

a=12

bewirkt Folgendes: Der Computer stellt einen Speicher bereit, welcher den Namen **a** bekommt (der Speichername steht immer links von =) und legt in diesen Speicher den Wert 12 ab (der Wert, der abgespeichert werden soll, steht immer rechts von =).
Mit **PRINT a** können wir uns den Inhalt des Speichers **a** anzeigen lassen.

19) Gib im Direktmodus ein:
x = 3 (EINGABE-Taste)
PRINT x; 2*x; z (EINGABE-Taste)
Es werden richtig 3 und 6 angezeigt, dann kommt die Ausgabe 0. Speicher, in die bisher nichts gelegt wurde, enthalten in BASIC den Wert 0.

Anmerkungen:

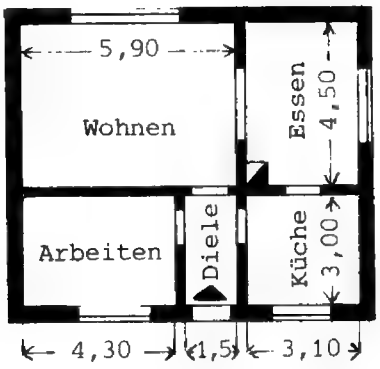
1. Man liest den Befehl **a=12** ausführlich als "dem Speicher **a** wird der Wert 12 zugewiesen" oder kurz: "a wird 12".
2. Statt vom "Speicher mit dem Namen a" redet man kurz von der *Variablen* **a**.
3. Wie du eben gesehen hast, ist es in BASIC möglich, dass Speicher, die noch nicht angelegt wurden, aufgerufen werden können. (Sie enthalten dann den Wert 0.) In allen modernen Programmiersprachen ist es notwendig, neue Variable zu 'vereinbaren' oder zumindest zu 'initialisieren', d.h. ihnen zuerst einen Wert zuzuweisen, bevor sie angesprochen werden können. Wir werden daher in unserem Buch neue Variable stets durch eine explizite Wertezuweisung 'initialisieren' (auch wenn BASIC dies nicht verlangt).

Der Input-Befehl

Unser Programm ist sehr unpraktisch, wenn man die Summe (die Differenz, das Produkt, den Quotienten) vieler Zahlenpaare berechnen muss. Für jedes neue Paar müsste man zuerst die Programmzeilen 3 und 4 ändern. Um dies überflüssig zu machen, gibt es einen eigenen Eingabebefehl.

Ein Bodenleger muss den Flächeninhalt der einzelnen Zimmer berechnen. Er schreibt sich dieses Programm:

```
' Programm des Bodenlegers
CLS
INPUT laenge
INPUT breite
flaeche=laenge*breite
PRINT flaeche
END
```



20) Lösche dein altes Programm: Entweder markieren und Löschen mit der ENTF-Taste oder **Datei Neu** (Abfrage zum Speichern verneinen). Gib nun das Programm des Bodenlegers ein und starte es mit F5.

Auf dem Bildschirm erscheint ein "?". Der Computer zeigt dir dadurch an, dass er eine Eingabe von dir erwartet. Lies also aus dem Grundriss den Wert 5.90 für die Länge des Wohnzimmers, gib die Zahl ein und bestätige mit RETURN. Gleich erscheint das Fragezeichen wieder. Der Computer möchte ja noch den Wert für die Breite von dir wissen. Gib also den entsprechenden Wert 4.50 aus dem Grundriss ein. Der Computer zeigt nun das Produkt der eingegebenen Zahlen an. Nach dem ersten Lauf des Programmes müsste dein Bildschirm so aussehen:

```
? 5.90
? 4.50
26.55
```

Der INPUT-Befehl bietet eine weitere Form der Wertzuweisung. Sie erfolgt während des Programmlaufs: Das Programm wird angehalten, bis der Benutzer seinen Wert eingegeben und die Eingabe mit der EINGABE-Taste bestätigt hat. Der eingegebene Wert wird der Variablen, deren Namen hinter INPUT steht, zugewiesen. ***Kommazahlen dabei stets mit Dezimalpunkt eingeben.***

21) Lass dir im Direktmodus den Inhalt der Variablen **laenge**, **breite** und **flaeche** ausgeben.

Unser Bodenleger lässt seine Auszubildende Vera die Flächeninhalte aus dem Grundriss berechnen. Da Vera noch nicht mit Programmen gearbeitet hat, schreibt er sein Programm für sie um:

```
' Verbessertes Programm des Bodenlegers
CLS
PRINT "     Flächeninhalte"
PRINT
INPUT "Länge des Zimmers in Meter = ",laenge
INPUT "Breite des Zimmers in Meter = ",breite
flaeche=laenge*breite
PRINT
PRINT "Flächeninhalt in Quadratmeter = ";flaeche
END
```

Der Befehl in Zeile 2 löscht den Bildschirm. Zeile 3 schreibt eine Überschrift, die Zeilen 4 und 8 bewirken die Ausgabe einer Leerzeile. Zeile 5 und 6: Anstelle des Fragezeichens erscheint jetzt der bei **INPUT** stehende Text und erläutert die gewünschte Eingabe. Die "Syntax" des **INPUT**-Befehls verlangt, dass nach dem Text und vor der Variablen ein Komma steht. Zeile 9 druckt das Rechenergebnis mit einer Erläuterung.
Beachte: Variablennamen können in Q-BASIC bis zu 40 Zeichen enthalten, sie müssen mit einem Buchstaben beginnen. Deutsche Sonderzeichen wie 'ä', 'ö', 'ü', 'ß' sind nicht erlaubt. In einem Text (also zwischen" ...") dürfen diese Zeichen aber vorkommen.

22) Tippe obiges Programm ein und berechne mit seiner Hilfe die Flächeninhalte der übrigen Zimmer im Grundriss.

23) Unser Bodenleger benötigt auch Leisten für jedes Zimmer. Er will sein Programm so ergänzen, dass auch gleich die Länge der benötigten Bodenleisten für jedes Zimmer berechnet wird. (Türen berücksichtigt er dabei nicht, die deshalb zu viel bestellten Leisten werden erfahrungsgemäß gerade für den unvermeidlichen Verschnitt gebraucht.) Ergänze das letzte Programm entsprechend.

24) Heike soll ein Programm schreiben, das den Flächeninhalt von Quadraten berechnet.

Lösche das Programm zu Aufgabe 23. Gib dann Heikes Programm ein und berechne den Flächeninhalt einiger Quadrate.

25) a) Ergänze Heikes Programm um diese Zeilen:

```
CLS
PRINT "Flächeninhalt und Umfang von Quadraten"
umfang=4*seite
PRINT "Umfang =";umfang
END
```

Gib diese Zeilen an den passenden Stellen ein und starte das Programm.
b) Ergänze Heikes Programm noch um eine Programmüberschrift und sorge dafür, dass der Bildschirm durch Leerzeilen übersichtlich gestaltet wird.

26) Dein Programm gefällt dir jetzt so gut, dass du es auf Diskette speichern möchtest. Wähle im Menü **Datei** den Punkt **Speichern unter...** und gib Laufwerk, Pfad und Name des Programms ein. *Beispiel*: Lege auf deiner Diskette im Laufwerk A: ein Verzeichnis QBASIC und in diesem das Unterverzeichnis KAP1 an. (Dies machst du am Besten unter Windows Dateimanager). Dann wählst du durch Anklicken Laufwerk A: und wieder durch zweimaliges Anklicken das entsprechende Verzeichnis, schließlich tippst du den Namen, meinetwegen AUFG25 ein und bestätigst mit OK.

27) Ein Quadrat hat den Flächeninhalt 20 cm^2. Wie groß ist seine Seitenlänge? Lege eine solche Tabelle an und bestimme die gesuchte Seitenlänge auf Zehntelmillimeter genau durch systematisches Probieren mit deinem Programm.

Seitenlänge (cm)	4	5	4.5	4.4	.
Flächeninhalt (cm^2)	16	25	20.25	.	.

Beim Abschreiben der Werte stört der Ausdruck des Umfangs. Durch Markieren und Löschen entfernst du die entsprechenden Zeilen. Lösche aus dem Programm "**quadrat**" alle Zeilen, die sich auf den Umfang beziehen.

So gehst du bei der Eingabe von Programmen vor:

1. Datei Neu - Der Bildschirm wird frei gemacht (falls nötig).
2. Programm Zeile für Zeile eingeben.
3. SHIFT F5 - Das Programm wird ausgeführt.
4. Du kannst nachträglich Zeilen hinzufügen
5. Du kannst Zeilen löschen: Markieren und ENTF-Taste.
6. Du kannst eine Zeile ändern, indem du sie neu schreibst oder korrigierst.

2 Rechnen mit Q-BASIC

2.1 Lineare Programme

Kaufhaus Wöniger feiert sein 25-jähriges Jubiläum und kündigt einen Jubiläumsverkauf an; alle Artikel werden um 25% verbilligt. Lehrling Ludwig soll die neuen Preise auszeichnen. Er schreibt sich ein Programm.

```
'Ludwigs Programm
p=25 ' Prozentsatz in Prozent
PRINT
INPUT "Alter Preis? ", preis
rabatt = preis*p/100
neupreis = preis-rabatt
PRINT "Neuer Preis: ";neupreis
```

Mit **INPUT** wird die Eingabe, mit **PRINT** die Ausgabe bewerkstelligt. Die Befehle in den Zeilen 2, 5 und 6 heißen *Wertzuweisungen*. In Zeile 2 ist dies am deutlichsten: Der *Variablen* p wird der *Wert* 25 zugewiesen. In den Zeilen 5 und 6 wird zunächst der Wert der *rechten* Seite berechnet und dann der *links stehenden* Variablen zugewiesen.

1) Tippe das Programm ein und starte es. Rechne die neuen Preise aus.

2) Lösche die Zeile 2 und starte das Programm. Was geschieht?

3) Ergänze 1) am Anfang durch die Zeile

```
CLS
```

Welche Verbesserung ergibt sich in der Ausgabe?

Mit Hilfe von Wertzuweisungen, **INPUT**- und **PRINT**-Befehlen kannst du schon viele einfache Programme schreiben, die alle nach demselben Muster ablaufen:

- *Eingabe* von Daten (mit **INPUT**)
- *Verarbeitung* von Daten (mit Grundrechenarten und Wertzuweisungen)
- *Ausgabe* von Daten (mit **PRINT**)

4) Nach Eingabe von Länge und Breite eines Rechtecks soll die Fläche ausgegeben werden.

5) Nach Eingabe von Länge und Breite eines Rechtecks soll der Umfang ausgegeben werden.

6) Die Mehrwertsteuer beträgt gewöhnlich 15% des Nettopreises. Es ist Bruttopreis = Nettopreis + Mehrwertsteuer. Schreibe ein Programm, das aus dem Nettopreis den Bruttopreis berechnet.

7) Ein Gelegenheitsarbeiter verdient 18,50 DM pro Stunde. Aus der Anzahl der Stunden soll der Verdienst berechnet werden.

8) Anton schreibt 3 Mathematik-Klassenarbeiten. Er wünscht sich ein Programm, das ihm blitzschnell den Notendurchschnitt berechnet.

9) Berta bekommt bei ihrem Sparbuch 3% Zinsen. Nach Eingabe des Kapitals soll der Computer den Jahreszins ausgeben.

10) Berta möchte ihr Geld besser anlegen und braucht deshalb ein allgemeines Programm: Auch der Zinssatz soll jetzt eingegeben werden.

11) Berta möchte auch noch ihr Kapital nach einem Jahr erfahren.

Die Wertzuweisung, genauer betrachtet

Für die bisherigen Programme genügte ein oberflächliches Verständnis der Wertzuweisung. Um dein "Schubladendenken" etwas zu schulen, versuche jetzt mal die Wirkung der folgenden Testprogramme vorauszusagen, bevor du sie eintippst und laufenlässt.

```
' Üb. 12
 x=3
 x=4
 PRINT x
```

```
' Üb. 13
x=3
y=4
x=y
PRINT x
```

```
' Üb. 14
x=3
x=x+1
PRINT x
```

Am schwierigsten ist Zeile 3 aus Übung 14 zu verstehen, hier werden aber auch die Unterschiede zum mathematischen Gleichheitszeichen am deutlichsten:

- Die Wertzuweisung arbeitet *von rechts nach links*. Der Wert des rechten Ausdrucks wird berechnet (falls nötig) und der Variablen, die links steht, zugewiesen.
- Die Wertzuweisung arbeitet *dynamisch* (im Gegensatz zum statischen Gleichheitszeichen), d.h. im Verlauf eines Programms nimmt eine Variable oft mehrere Werte nacheinander an.

Struktogramme

Struktogramme sind ein Hilfsmittel, um die Struktur eines Programmes deutlich zu machen. Obwohl dies bei den bisherigen Programmen nicht nötig war, wollen wir am Ende dieses Abschnitts ein Beispiel vorstellen. Es sei die Aufgabe gestellt, aus Länge, Breite und Höhe eines Quaders die Oberfläche zu berechnen.

Das nebenstehende Struktogramm könnte entstanden sein als Versuch eines Programmierers, seine Gedanken zu ordnen, oder als Hilfe eines Lehrers für einen Programmierneuling.

Eingabe: **Länge l** **Breite b** **Höhe h**
Oberfläche = 2*(l*b+b*h+h*l)
Ausgabe: **Oberfläche**

15) Schreibe das zugehörige Programm.

Das Struktogramm soll die wesentlichen Gedanken eines Programmes festhalten, ohne auf die starren Regeln einer Programmiersprache einzugehen. Es kann uns das Programmieren erleichtern; dies wird aber erst bei größeren Programmen deutlich.

2.2 Wiederholungen

Der Computer kann gleichartige Vorgänge sehr schnell wiederholen. Eine Möglichkeit, ihn dazu zu veranlassen, wollen wir jetzt kennen lernen.

Die gezählte Wiederholung

Lehrling Ludwig möchte sein Programm aus Übung 1 nicht immer wieder neu starten. Er schreibt deshalb:

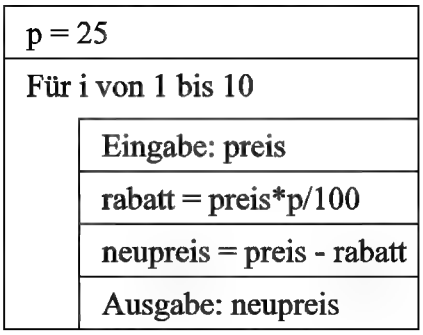
16)
```
' Ludwigs Wiederholung
p=25
FOR i=1 TO 10
  INPUT "Preis? ", preis
  rabatt = preis*p/100
  neupreis = preis-rabatt
  PRINT "Neupreis = ",neupreis
NEXT i
```

Zwischen den Schlüsselwörtern **FOR** und **NEXT** steht der *Schleifenrumpf*, das ist der Teil, der 10-mal wiederholt wird. Beim Schreiben des Programms rücken wir den Schleifenrumpf ein, um die Struktur des Programms zu verdeutlichen.

Die Variable i heißt Laufvariable oder Zählvariable. Sie erhält hier zunächst den Wert 1 und wird bei jedem Durchlaufen des Schleifenrumpfes um 1 erhöht bis der Wert 10 erreicht ist.

Wenn ein Sportler im Stadion seine Runden dreht, lässt er oft einen Freund mitzählen, in welcher Runde er sich jeweils befindet. Diese Rolle übernimmt hier die Zählvariable i.

Rechts kannst du sehen, wie die **FOR**-Schleife im Struktogramm dargestellt wird. Der Schleifenrumpf wird durch Einrücken her-vorgehoben.



17) Verbessere die Ausgabe durch einen **CLS**-Befehl und durch Leerzeilen.

18) Was geschieht, wenn der **INPUT**-Befehl außerhalb der Schleife steht ?

19) Ludwig will die Anzahl der Wiederholungen beim Programmlauf wählen können. Ändere das Programm entsprechend ab (das Programm soll die Anzahl mit **INPUT** erfragen).

20) Was leistet das nebenstehende Programm? Was ändert sich, wenn in Zeile 3 am Schluss ein Komma bzw. ein Strichpunkt gesetzt wird?

```
' Zählen
FOR k=1 TO 10
 PRINT k
NEXT k
```

21) Der Computer soll von 100 bis 120 zählen.

22) Die Zahlen von 1 bis n sollen geschrieben werden (n über **INPUT** erfragen).

23) Auf dem Bildschirm sollen 40 Sternchen erscheinen.

24) Es sollen die Quadratzahlen 1, 4, 9, . . . 100 ausgegeben werden.

25) Erzeuge eine Tabelle der Quadratzahlen.

```
1    1
2    4
3    9
4    16
```

26) Lasse den Benutzer entscheiden, wie weit die Quadratzahltabelle gehen soll.

27) Die Vielfachen von 7 sollen ausgegeben werden 7, 14, 21, . . . 70

28) Eines der folgenden Bilder soll auf dem (Text-)Bildschirm entstehen. Hinweis: Arbeite mit zwei geschachtelten FOR-Schleifen.

```
*****        *        Hallo
*****        **        Hallo
*****        ***        Hallo
*****        ****        Hallo
```

Der kleine Carl-Friedrich Gauß soll seinen Lehrer verblüfft haben, indem er die Summe aller Zahlen von 1 bis 100 blitzschnell errechnete. Wir wollen zunächst kleiner anfangen:

29) Der Computer soll die Zahlen von 1 bis 10 addieren. Die Lösung sei hier gleich angegeben, sie verwendet den dynamischen Charakter der Wertzuweisung.

```
' Summe von 1 bis 10
PRINT "Summe der Zahlen 1 bis 10"
summe=0
FOR i=1 TO 10
  summe=summe+i' alte Summe+i gibt neue Summe
NEXT i
PRINT "Die Summe ist";summe
```

30) Was passiert, wenn Zeile 3 vergessen wird?

31) Schreibe ans Ende des Schleifenrumpfes die Zeile

```
PRINT summe
```

32) Verallgemeinere auf die Summe der Zahlen von 1 bis n (n über **INPUT** erfragen).

33) Otto hat 4 Klassenarbeiten geschrieben und möchte ein Programm, das den Notendurchschnitt berechnet. Er benützt eine FOR-Schleife.

```
' Notendurchschnitt
summe=0
FOR i=1 TO 4
  INPUT "Note? ", note
  summe=summe+note
NEXT i
durchschnitt=summe/4
PRINT "Durchschnitt: ";durchschnitt
```

34) Verallgemeinere auf n Klassenarbeiten (n über **INPUT** erfragen).

35) Berta hat 100 DM auf dem Sparkonto und erhält 3% Zinsen pro Jahr, die sie jeweils auf dem Konto belässt. Um einen Überblick über die Kapitalentwicklung zu bekommen, schreibt sie unten stehendes Programm:

```
' Kapitalentwicklung
anfangskapital=100
p=3 ' Zinssatz in Prozent
kapital=anfangskapital
FOR i=1 TO 10
```

```
    zins=kapital*p/100
    kapital=kapital+zins
    PRINT "Nach";i;"Jahren:";kapital;"DM"
NEXT i
```

Der neue Gedanke ist hier, dass nicht mehr wie in 11) *zwei* Variablen für Anfangs- bzw. Endkapital eingerichtet werden, denn das Endkapital eines Jahres ist das Anfangskapital des nächsten. Deshalb ist hier *eine* Variable **kapital** geschickter.

36) Wir wollen die Ausgabe in Tabellenform. Wir verändern dazu den PRINT-Befehl

```
PRINT i, kapital
```

Um eine verständliche Ausgabe zu erhalten, müssen wir auch noch eine Überschrift über der Tabelle erzeugen (siehe rechts). Schreibe das Programm entsprechend um.

```
Jahr            Kapital

1               103
2               106.09
3               109.2727
4               112.550881
5               115.927407
6               119.40523
7               122.987387
8               126.677008
9               130.477318
10              134.391638
```

Immer noch ist die Tabelle recht unbefriedigend, was die äußere Form betrifft. Das wird vor allem deutlich, wenn wir als Anfangskapital etwa 980 DM nehmen. Wir hätten gern, dass die Dezimalpunkte immer untereinander stehen und DM-Beträge mit zwei Stellen hinter dem Komma angegeben werden. Dies erreichen wir, indem wir den PRINT-Befehl durch das Schlüsselwort **USING** ergänzen. Der PRINT-Befehl könnte dann zum Beispiel so lauten:

```
PRINT USING "Kapital nach ## Jahren: ######.## DM"; i,kap
```

Durch diesen Befehl wird jetzt genau festgelegt, an welchen Stellen der Zeile die Zahlen zu schreiben sind. Das Zeichen # (Nummernzeichen oder auch Gartenzaun genannt) hält den Platz für die Ziffern frei. Für die Variable i sind hier zwei Ziffern, für die Variable k sind sechs Ziffern vor und zwei Ziffern nach dem Dezimalpunkt vorgesehen. Buchstaben und Leerzeichen werden genauso übernommen, wie in der Vorlage angegeben.

37) Ändere das Programm entsprechend.

38) Spiele und experimentiere mit **PRINT USING** im Direktmodus, z.B.

```
PRINT USING"1, #"; 2
PRINT USING"# und 2"; 3
PRINT USING"    #ung !"; 8
PRINT USING"#; zwei; ###"; 1,3
PRINT USING"####.## DM"; 7
PRINT USING"  #.## kg"; 99
```

Was geschieht, wenn zu wenig Platz vor bzw. hinter dem Dezimalpunkt frei gehalten wird?

39) Verallgemeinere Aufgabe 36 so, dass nun Kapital, Zinssatz und die Anzahl der Jahre über **INPUT** erfragt werden.

40) Probiere mit Hilfe von Aufgabe 39, nach wie vielen Jahren sich ein Kapital bei 3% (bzw. bei 7%) Zinsen verdoppelt hat.

Bedingte Wiederholungen

Die Lösung der vorigen Aufgabe war sicher unbefriedigend für dich, denn du musstest die Anzahl der Wiederholungen angeben, wusstest sie aber nicht. Da half nur Probieren.
Wir brauchen einen Befehl, der die folgende Form hat:

Solange sich das Kapital noch nicht verdoppelt hat,
führe den Schleifenrumpf aus.

In Q-BASIC lauten die entsprechenden Schlüsselwörter:

WHILE ...
...
WEND

Dabei dient das Wort **WEND** dazu, das Ende des Schleifenrumpfes anzuzeigen.

41) Sieh dir an, wie man das Problem der Kapitalverdoppelung mit einer **WHILE**-Schleife löst.

```
' Kapitalverdopplung
anfangskapital=100
p=3 ' Zinssatz in Prozent
i=0
kapital=anfangskapital
WHILE kapital<2*anfangskapital
  i=i+1 ' i zählt die Jahre
  zins=kapital*p/100
  kapital=kapital+zins
  PRINT "Kapital nach";i;"Jahren:";kapital;"DM"
WEND
PRINT
PRINT "Verdoppelung nach ";i;" Jahren."
```

42) Verallgemeinere das Problem mit Hilfe eines **INPUT**-Befehles so, dass der Zinssatz vom Benutzer eingegeben wird. Es gibt eine einfache Faustformel, wie die Verdoppelungszeit vom Zinssatz abhängt. Versuche sie zu finden, indem du das Programm mit verschiedenen Zinssätzen rechnen lässt.

43) Der Computer soll die Zahlen von 1 bis 100 ausgeben. Natürlich macht man das mit einer FOR-Schleife. Versuche es zur Übung mit einer WHILE-Schleife. Falls es dir nicht gelingt, lies dir durch, wie dieses Problem in Programm 41 gelöst wurde, wo die Jahre ausgegeben wurden.

Wir kehren nochmal zum Problem der Kapitalverdoppelung zurück. Anstatt

Solange sich das Kapital noch nicht verdoppelt hat, führe
den Schleifenrumpf aus.

kann man auch sagen:

Führe
den Schleifenrumpf aus,
bis sich das Kapital verdoppelt hat.

In Q-BASIC lauten die entsprechenden Schlüsselwörter:

DO

...

LOOP UNTIL . . .

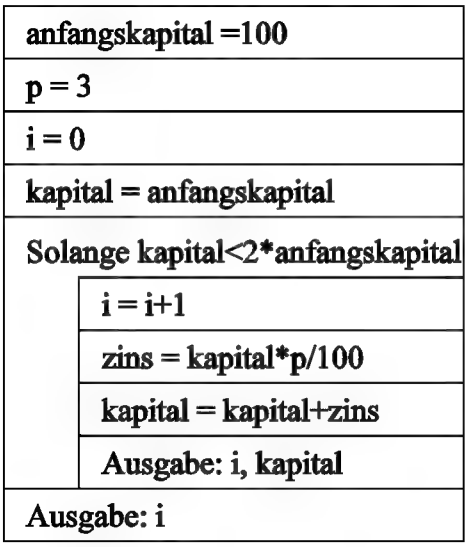
Zwischen **DO** und **LOOP UNTIL** steht der Schleifenrumpf, hinter **UNTIL** kommt die Bedingung, bei der abgebrochen werden soll.

44) Sieh dir an, wie man das Problem der Kapitalverdoppelung mit einer **UNTIL**-Schleife löst.

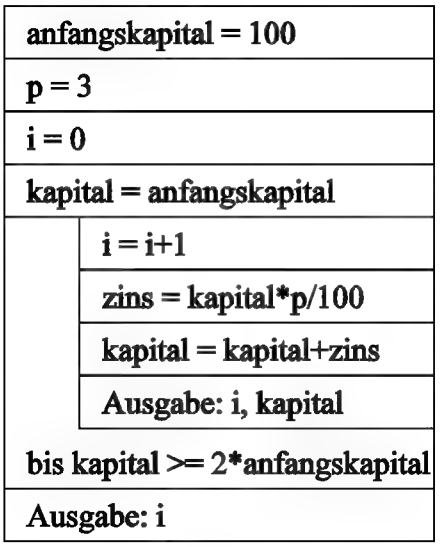
```
' Kapitalverdoppelung
anfangskapital=100
p=3 ' Zinssatz in Prozent
i=0
kapital=anfangskapital
DO
  i=i+1 ' i zählt die Jahre
  zins=kapital*p/100
  kapital=kapital+zins
  PRINT "Kapital nach";i;"Jahren:";kapital;"DM"
LOOP UNTIL kapital>=2*anfangskapital
PRINT
PRINT "Verdoppelung nach";i;"Jahren."
```

Vergleiche die Beispiele in 41) und 44). Die Struktogrammdarstellung der beiden Schleifenformen wollen wir anhand dieser beiden Beispiele einführen.

WHILE-Schleife



UNTIL-Schleife



Bei den Struktogrammen siehst du deutlich den Unterschied zwischen den beiden Schleifenformen:

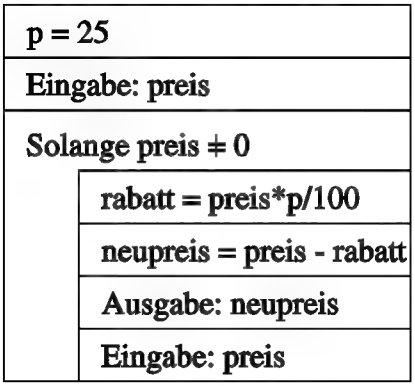
- bei der **WHILE**-Schleife wird *vor* Durchlaufen des Schleifenrumpfes eine *Laufbedingung* getestet.
- bei der **UNTIL**-Schleife wird *nach* Durchlaufen des Schleifenrumpfes eine *Abbruchbedingung* getestet.

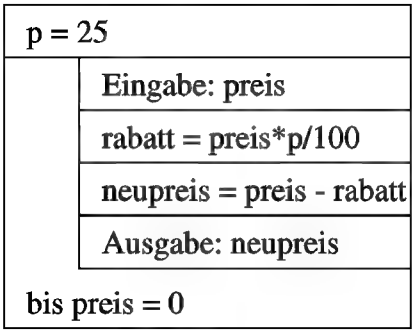
Vielleicht fragst du dich, warum wir für die bedingte Wiederholung zwei verschiedene Möglichkeiten anbieten. Das nächste Beispiel kann eine erste Antwort geben:

Lehrling Ludwig verbessert sein Programm (16) weiter. Da er zu Beginn des Programmlaufs noch gar nicht weiß, wie viele Preise er umrechnen muss, scheint ihm eine **WHILE**-Schleife oder eine **UNTIL**-

Schleife geeigneter als eine **FOR**-Schleife. Das Programm soll enden, wenn der Benutzer statt eines Preises eine Null eingibt.

Hier die beiden Möglichkeiten im Struktogramm:





Die rechte Lösung mit **UNTIL** ist einfacher zu verstehen. Da am Ende des Schleifenrumpfes der eingegebene Preis bekannt ist, kann er in der Abbruchbedingung abgefragt werden. Bei der **WHILE**-Schleife ergibt sich die Schwierigkeit, dass die Laufbedingung den eingegebenen Preis benötigt. Die erste Eingabe muss daher noch außerhalb des Schleifenrumpfes geschehen.

45) Schreibe zum rechten Struktogramm das zugehörige Programm mit einer **UNTIL**-Schleife.
46) Schreibe das entsprechende Programm mit der **WHILE**-Schleife.

Bauer Meier verpackt jeden Morgen die Eier seiner Hühner in Kartons zu 6 Stück. Er hat dafür ein Programm:

```
' Meiers Programm
i=0
INPUT "Wie viele Eier? ", n
DO
  i=i+1
  n=n-6
LOOP UNTIL n<6
PRINT "Man braucht";i;"Kartons."
PRINT "Es bleiben";n;"Eier übrig."
```

Meier war mit diesem Programm sehr zufrieden, bis eines Tages seine Hühner nur 5 Eier legten. Als er spaßeshalber das Programm laufen ließ, erschien eine unsinnige Ausgabe. Begründe, warum das Programm in diesem Fall "spinnt".

47) Verbessere Meiers Programm mit einer **WHILE**-Schleife, sodass auch bei einer Eierzahl unter 6 das Programm die richtige Lösung findet.

48) Der Computer soll alle geraden Zahlen von 2 bis 100 schreiben.

49) Der Computer soll rückwärts zählen: 100, 99, . . . , 1.

FOR-Schleife mit STEP

Die letzten Aufgaben erinnern stark an eine **FOR**-Schleife, obwohl sie nicht unmittelbar mit einer solchen gelöst werden können. Für diese und ähnliche Aufgaben gibt es das Schlüsselwort **STEP**. Damit gibt man an, welche Schrittweite anstatt 1 gewählt wird. Zum Beispiel für die vorige Aufgabe:

```
' Rückwärtszählen
FOR i=100 TO 1 STEP -1
  PRINT i,
NEXT i
```

50) Der Computer soll alle geraden Zahlen von 2 bis 100 schreiben.

51) Alle ungeraden Zahlen von 1 bis 99 sollen geschrieben werden.

52) Es soll eine Tabelle der unten stehenden Form geschrieben werden. Drucke die Tabelle auch mit den Schrittweiten 0.1 bzw. 0.25. Verbessere die Ausgabe mit PRINT USING.

```
x           x^2

-1          1
-.8         .64
-.6         .36
```

usw.

53) Experimentiere mit dem unten stehenden Programm. Was geschieht, wenn Anfang, Ende und Schrittweite nicht zusammenpassen?

```
' Test
INPUT "Anfang? ", anfang
INPUT "Ende? ", ende
INPUT "Schrittweite? ", s
FOR x=anfang TO ende STEP s
  PRINT x,
NEXT  x
```

54) Wie groß ist die Summe aller ungeraden Zahlen, die kleiner oder gleich 11 sind? Wie groß ist die Summe aller ungeraden Zahlen, die kleiner gleich der Zahl n sind? Probiere mit dem Computer!

Weitere Schleifenformen

Mit der FOR-Schleife und der WHILE-Schleife hast du die beiden wichtigsten Schleifenformen kennen gelernt. Diese Schleifenformen gibt es auch in allen verwandten Sprachen (PASCAL, C usw.). Darüber hinaus gibt es in Q-BASIC noch eine sehr allgemeine Schleifenform, die DO-LOOP-Schleife. Wir haben sie schon als UNTIL-Schleife verwendet. Darüber hinaus kann man die DO-LOOP-Schleife auch mit dem Befehl EXIT DO verwenden, der einen Sprung aus der DO-LOOP-Schleife bewirkt. Diese Form werden wir im Folgenden nicht verwenden, sie kann aber bei manchen Problemen das Programm eleganter (oft auf Kosten der Übersichtlichkeit) machen.

Falls dich die DO-LOOP-Schleife interessiert, kannst du die hier eine andere Lösung der Aufgabe 45 bzw. 46 anschauen.

```
'Ludwigs Programm mit DO LOOP
p=25
DO
  INPUT "Preis? ", preis
  IF preis=0 THEN EXIT DO
  rabatt = preis*p/100
  neupreis = preis-rabatt
  PRINT "Neupreis:";neupreis,
  PRINT
LOOP
```

WHILE-Schleife und DO-LOOP-Schleife sind gleichwertig, d.h. eine der beiden Formen kann immer durch die andere ersetzt werden. Die WHILE-Schleife orientiert sich mehr an unserer Denkstruktur, die DO-LOOP-Schleife mehr an der Abarbeitung durch die Maschine.

Wir werden in Zukunft die DO-LOOP-Schleife nur in der Form der UNTIL-Schleife verwenden.

2.3 Entscheidungen

Bei Rechenaufgaben müssen wir oft Entscheidungen treffen. Ein Beispiel:
Die monatlichen Telefongebühren der Familie Müller werden von der Post folgendermaßen berechnet: Die Grundgebühr beträgt 27 DM, die ersten 10 Einheiten sind umsonst, ab dann sind 23 Pfg pro Einheit zu bezahlen. Bezeichnen wir die Anzahl der Einheiten mit n, so müssen wir *zwei Fälle* unterscheiden: $n \leq 10$ bzw. $n > 10$.

Wenn $n \leq 10$ ist, **dann**
betragen die Gebühren 27,00 DM
sonst
betragen die Gebühren 27,00 DM + (n-10)*0,23 DM.

Dies kann man in Q-BASIC ausdrücken mit

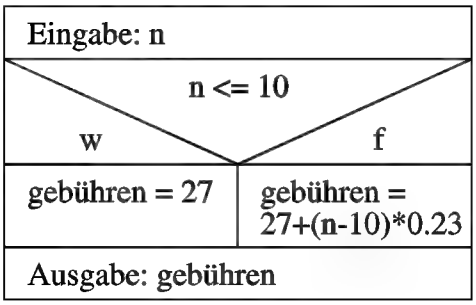
IF ... **THEN**
.....
ELSE
.....
END IF

Die Schlüsselwörter **END IF** zeigen an, dass die Fallunterscheidung beendet ist.

55) Unser obiges Beispiel sieht als Programm dann so aus:

```
'Telefongebühren
INPUT "Anzahl der Einheiten? ", n
IF n<=10 THEN
 gebuehren=27
ELSE
  gebuehren=27+(n-10)*.23
END IF
PRINT "Die Gebühren betragen";gebuehren;"DM"
```

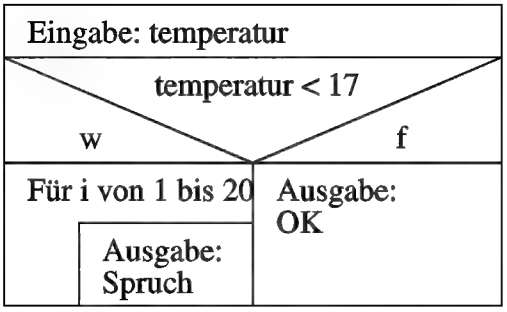
Je nachdem, ob die Bedingung erfüllt ist oder nicht, wird die eine oder die andere Anweisung ausgeführt. Man spricht von einer Verzweigung des Programms. Diese wird im Struktogramm wie nebenstehend dargestellt. Dabei steht "w" für "wahr" und "f" für "falsch".



56) Emil hat zum zwölften Geburtstag einen Computer bekommen. Er schreibt ein Programm für seine Freunde. Diese müssen ihr Alter eingeben. Ist dieses kleiner als 12, so erscheint "Hallo Baby", sonst "Hallo Kumpel".

57) Beim Trimtrab sollte der Puls nicht mehr als 130 betragen. Gesucht ist ein Programm, das die Pulszahl erfragt und dann gute Ratschläge erteilt.

58) Frau Müller ist sehr besorgt um ihren Sohn. Wenn es morgens kühl ist, ermahnt sie ihn: "Robi, vergiss deine Jacke nicht!" Robert schreibt ihr ein Programm, bei dem sie die Temperatur eingeben muss. Ist diese kleiner als 17 Grad, erscheint besagter Spruch zwanzigmal auf dem Bildschirm, sonst erscheint ein ´OK´.



Abkürzende Schreibweisen

Falls Robert bei der vorigen Aufgabe will, dass nur dann eine Meldung erscheint, wenn die Temperatur kleiner als 17 Grad ist, so brauchen wir das Schlüsselwort ELSE nicht zu schreiben. Dies entspricht ganz genau unserer üblichen Sprechweise in der Umgangssprache. Das Programm lautet dann kürzer:

```
INPUT "Temperatur", temperatur
IF temperatur<17 THEN
 FOR i=1 TO 20
  PRINT"Robi, vergiss deine Jacke nicht!"
 NEXT i
END IF
```

Falls die Temperatur mindestens 17 Grad beträgt, macht der Computer hier nichts mehr.

Diese Schreibweise können wir immer dann verwenden, wenn es sich nicht um eine Verzweigung im eigentlichen Sinn handelt, d.h. wenn nichts getan werden soll, falls die Bedingung hinter **IF** nicht erfüllt ist.

Diese Kurzschreibweise kann noch weiter verkürzt werden, falls die auszuführende Anweisung in eine Zeile paßt. Nehmen wir an, im obigen Beispiel wollten wir den Spruch „Robi, vergiss deine Jacke nicht“ nur einmal ausgeben. Dann könnten wir dies auf zwei gleichwertige Arten tun: Die eine Schreibweise haben wir soeben kennen gelernt:

```
INPUT "Temperatur", temperatur
IF temperatur<17 THEN
 PRINT"Robi, vergiss deine Jacke nicht! "
END IF
```

Die andere Möglichkeit ist noch kürzer:

```
INPUT "Temperatur", temperatur
IF temperatur<17 THEN PRINT"Robi, vergiss deine Jacke nicht! "
```

Während bisher jedes **IF** durch ein **END IF** abgeschlosssen wurde, ist das bei dieser Schreibweise nicht mehr der Fall. Trotzdem fällt uns diese Kurzschreibweise nicht schwer, da sie ganz unserer umgangssprachlichen Formulierung entspricht. Die kurze Schreibweise kann Programme übersichtlicher machen; versuche aber nicht, zu viel in eine Zeile zu packen.

Noch mehr Entscheidungen

Wenn in einem Programm mehrere Entscheidungen vorkommen, verliert man leicht den Überblick. Ein Struktogramm ist oft hilfreich.

59) Das Trimtrabprogramm aus Aufgabe 57 ist noch zu einfältig, verbessere es: Für den Fall, dass der Puls < 130 ist, soll noch unterschieden werden, ob der Traber eine Pulszahl < 100 hat, sich also zu wenig anstrengt.

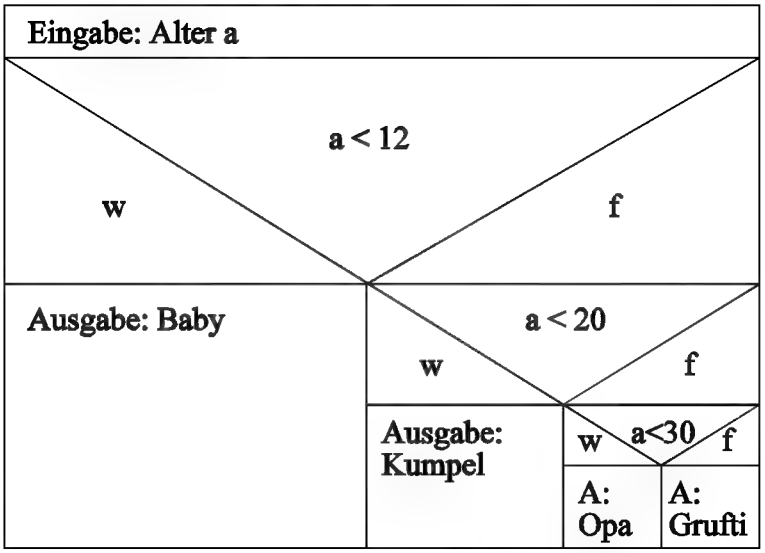
60) Auch das Baby-Programm von Aufgabe 56 kannst du noch erweitern: Falls der Benutzer über 19 Jahre alt ist, soll "Hallo Opa" ausgegeben werden.

61) Eine weitere "Verbesserung" dieses Programms liegt auf der Hand: ab 30 "Grufti". Das Programm wird dann sehr verschachtelt:

als Programm

```
INPUT "Alter?", a
IF a<12 THEN
  PRINT "Baby"
ELSE
  IF a<20 THEN
    PRINT "Kumpel"
  ELSE
    IF a<30 THEN
      PRINT "Opa"
    ELSE
      PRINT "Grufti"
    END IF
  END IF
END IF
```

als Struktogramm:



62) Mache so weiter: ab 40 soll "Obergrufti" erscheinen.

Die obigen Verschachtelungen sind etwas lästig. Als Abhilfe bietet Q-BASIC das Schlüsselwort **ELSEIF** an (was nichts anderes bedeuten soll als **ELSE IF**). Das Beispiel 61 lautet dann:

```
INPUT "Alter? ",a
IF a<12 THEN
  PRINT "Baby"
ELSEIF a<20 THEN
  PRINT "Kumpel"
ELSEIF a<30 THEN
  PRINT "Opa"
ELSE
  PRINT "Grufti"
END IF
```

Das Programm leistet genau dasselbe wie 61), ist aber übersichtlicher. Die Bedingungen werden von oben nach unten nachgeprüft. Sobald eine Bedingung zutrifft, wird die zugehörige Anweisung ausgeführt und danach wird das Programm hinter **END IF** fortgesetzt.

63) Schreibe Aufgabe 62 mit Hilfe von **ELSEIF** übersichtlicher.

64) Das Trimtrab-Programm soll weiter verbessert werden. Die Ausgabe soll sich laut Tabelle folgendermaßen nach der eingegebenen Pulszahl p richten. Vorsicht: Vergleiche mit Aufgabe 63.

	Ausgabe
$p < 50$	Suchen Sie Ihren Arzt auf
$50 \leq p < 100$	Sie können sich noch mehr anstrengen
$100 \leq p < 140$	Sie verhalten sich richtig
$140 \leq p < 200$	Sie strengen sich zu sehr an
$200 \leq p$	Ihre Uhr geht zu langsam

65) (nur als Ergänzung) Hier ist ein einfaches Programm, das die Reaktionszeit des Benutzers misst (einige Schlüsselwörter in diesem Programm werden erst weiter hinten im Buch erläutert). Ergänze das Programm durch einen Teil, der einen Kommentar zur Reaktionszeit ausgibt ("lahme Ente", "fixes Bürschchen" usw.).

```
'Messung der Reaktionszeit
CLS
PRINT "Drücke irgendeine Taste, sobald das Wort 'Los' erscheint"
PRINT
RANDOMIZE TIMER
SLEEP 3 * RND
PRINT "Fertig"
PRINT
SLEEP 3 * RND+1
PRINT "Los"

startzeit = TIMER
SLEEP
reaktionszeit = TIMER - startzeit
PRINT "Deine Reaktionszeit: "; reaktionszeit
```

Du brauchst nicht zu versuchen, das obige Programm in allen Einzelheiten zu verstehen, es genügt, dass du weißt, dass die Reaktionszeit bestimmt wird. Jetzt schreibst du einen Programmteil, der den Inhalt der Variable **reaktionszeit** verwendet.

Falls dich das Programm aber interessiert, kannst du dir mit der **Hilfe** aus dem Menü die Schlüsselwörter anschauen, die du noch nicht kennst: SLEEP, TIMER, RND, RANDOMIZE und INKEY$. Diese Anweisungen und Funktionen sind beim Programmieren von Spielen unentbehrlich. Ihre Beschreibung in der Hilfe ist nicht allzu schwierig. Auch im Anhang sind diese Schlüsselwörter kurz erklärt.

Zufallszahlen

Der Computer stellt uns so genannte Zufallszahlen zur Verfügung. Gib im Direktmodus mehrmals hintereinander ein:

```
PRINT RND
```

(Das Schlüsselwort **RND** ist eine Abkürzung für random, hier in der Bedeutung von zufällig.)

Du siehst: Es werden Zahlen zwischen 0 und 1 ausgegeben, eine Gesetzmäßigkeit kann man dabei nicht erkennen, in diesem Sinn sind die Zahlen also zufällig. In Wirklichkeit muss der Computer natürlich diese Zahlen berechnen, vorsichtige Leute sprechen deshalb lieber von Pseudozufallszahlen.

66) Schreibe ein Programm, das 10 Zufallszahlen ausgibt. Lasse dieses Programm mehrmals hintereinander ablaufen. Was fällt dir auf?

67) Um dem beobachteten Missstand abzuhelfen, schreiben wir an den Anfang eines Programms, das Zufallszahlen verwendet, stets die Zeile

RANDOMIZE TIMER

Mache das beim obigen Programm und beobachte die Wirkung.

68) Vielleicht hast du schon einmal eine Münze geworfen, um eine zufällige Entscheidung zwischen zwei Möglichkeiten herbeizuführen. Diesen Vorgang können wir auf dem Computer simulieren, d.h. nachahmen. Wenn die Zufallszahlen wirklich zufällig sind, ist die Wahrscheinlichkeit, eine Zufallszahl kleiner als 0,5 zu bekommen gerade 50%. Wir schreiben deshalb

```
RANDOMIZE TIMER
zufall = RND
IF zufall<0.5 THEN
 PRINT "Zahl"
ELSE
 PRINT "Wappen"
END IF
```

69) Schreibe ein Programm, das zehnmal eine Münze wirft. Lasse das Programm mehrmals laufen und beobachte die Häufigkeit von Wappen.

70) Schreibe ein Programm, das solange eine Münze wirft, bis Wappen fällt. Lasse das Programm mehrmals laufen und beobachte das Verhalten der Münze.

71) Bette das vorige Programm in eine FOR-Schleife ein, damit der Vorgang selbsttätig wiederholt wird.

72) Im Prinzip könnten wir auch das Werfen eines Würfels auf diese Art simulieren. Üblicherweise wird das aber anders gemacht. Schreibe das folgende Programm ab, und versuche es zu verstehen:

```
RANDOMIZE TIMER
zufall = INT(6*RND)+1
PRINT zufall
```

Das Programm erzeugt eine Zufallszahl zwischen 1 und 6, simuliert also das Würfeln. Um das Programm zu verstehen, musst du wissen, was das Schlüsselwort **INT** bedeutet: Es bewirkt, dass bei einer Zahl die Stellen hinter dem Dezimalpunkt weggelassen werden.

73) Simuliere das 60-malige Werfen eines Würfels. Wie oft fällt eine Sechs? Kannst du das Programm so erweitern, dass die Anzahl der Sechsen mitgezählt und ausgegeben wird?

74) Beim Spiel "Mensch ärgere dich nicht" darf man erst beginnen, wenn man eine 6 gewürfelt hat. Wie oft muss man dazu im Durchschnitt würfeln? Du hast sicher eine Vermutung. Überprüfe sie, indem du den Vorgang 100 (oder 1000) mal wiederholst und dann den Durchschnitt bildest.

75) Fred hat schon 10 mal gewürfelt und immer noch keine 6. "Das kommt öfters vor", trösten ihn die anderen, Fred aber hält sein Pech für einzigartig. Schätze durch Simulation ab, wie häufig es vorkommt, dass 10 oder noch mehr Würfe benötigt werden, bis eine 6 kommt.

Die folgenden Aufgaben zur Simulation dienen zur Vertiefung. Sie können zunächst übersprungen werden.

76) Anton, Berta und Clemens spielen ein Würfelspiel. Zwei Würfel werden geworfen; ist die Augensumme 2, 3, 4 oder 5, so gewinnt Anton, bei 6, 7, 8 gewinnt Berta, bei 9 , 10, 11 oder 12 gewinnt Clemens. Der Gewinner erhält von den beiden anderen Spielern je eine Mark. Wer hat wohl die besten Chancen? Ermittle die Gewinne und Verluste der drei Spieler bei 100 Spielen.

77) Max und Moritz spielen das Spiel *Craps*. Max hält die Bank, Moritz wirft zwei Würfel. Ist die Augensumme 7 oder 11, so hat er gewonnen, bei 2, 3 oder 12 hat er verloren. Ergibt sich eine andere Summe, nennen wir sie p, so muss er so lange weiterwürfeln, bis er 7 oder p als Augensumme erhält. Im Fall der 7 hat er verloren, im Fall p gewonnen. Schätze die Gewinnchancen von Moritz.

78) Peter und Paul spielen das Würfelspiel *Die verflixte Eins*. Abwechselnd darf jeder von ihnen eine beliebige Anzahl von Würfeln wählen und mit diesen gleichzeitig würfeln. Zeigt mindestens einer der Würfel eine 1, so erhält der Spieler nichts, andernfalls erhält er die Augensumme in Pfennigen von seinem Gegner.

    Peter ist sich unsicher. Nimmt er viele Würfel, so ist mit hoher Wahrscheinlichkeit eine 1 dabei, nimmt er wenige, oder gar nur einen Würfel, so erreicht er keine hohen Augensummen. Um schlauer zu werden simuliert Peter ein Spiel auf dem Computer. Nach Eingabe der Anzahl der Würfel soll der Computer 100-mal würfeln und den durchschnittlichen Gewinn ausgeben. Peter kann dann ausprobieren, welche Anzahl wohl am günstigsten ist.

    Um sich das Leben noch einfacher zu machen, lässt sich Peter das Ergebnis in Form einer Tabelle ausdrucken. Da er es genauer wissen will, lässt er jeden Versuch jetzt 1000-mal wiederholen, das dauert natürlich länger.

79) Der Besitzer eines Hotels mit 40 Zimmern weiß aus langjähriger Erfahrung, dass im Durchschnitt 10% der angemeldeten Gäste kurzfristig absagen. Deshalb nimmt er in der Hochsaison 42 Anmeldungen (also 5% zu viel) entgegen. War der Hotelier zu kühn? In wie viel Prozent aller Fälle kommt er in die peinliche Lage, angemeldete Gäste wieder fortschicken zu müssen?

    Die zufällige Entscheidung eines Gastes simulieren wir durch eine Zufallszahl zwischen 0 und 9, dabei soll 0 die kurzfristige Absage bedeuten. Wir lassen 42 Gäste entscheiden und zählen, wieviele ins Hotel kommen. Diesen Vorgang lassen wir oft wiederholen, um die obige Frage beantworten zu können.

80) Der geschäftstüchtige Hotelier vergrößert sein Hotel auf 80 Zimmer. Er sagt 84 Gästen (also wieder 5% zu viel) ein Zimmer zu. Simuliere auch diesen Vorgang und vergleiche.

2.4 Funktionen und Prozeduren

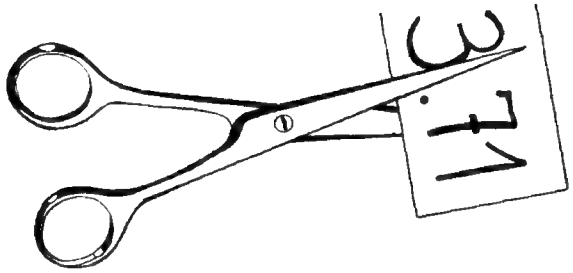
Die INT-Funktion

Berta hat ein Programm zur Berechnung des Jahreszinses geschrieben (siehe 10).

```
INPUT "Kapital? ", kap
INPUT "Zinsfuß? ", p
zins=kap*p/100
PRINT "Jahreszins:";zins
```

Ihre Schwester Dora, die bei einer Bank arbeitet, findet es unrealistisch, da in Wirklichkeit nur volle Markbeträge verzinst werden.
Berta grübelt: Wie kann man die Zahl kap *abrunden*? In Q-BASIC gibt es zu diesem Zweck die **INT**-Funktion (sprich: Integer-Funktion).

Ist x eine positive Zahl, so ist **INT(x)** diejenige Zahl, die beim Abrunden von x entsteht. Oder anders gesprochen:
INT schneidet von positiven Zahlen den Teil weg, der nach dem Dezimalpunkt kommt. Versuche es mal im Direktmodus:



```
PRINT INT(3.71)
PRINT INT(6.3)      usw.
```

Mit Hilfe dieser Funktion **INT** wird Bertas Programm wirklichkeitsnäher. Sie hat für die Zinsberechnung das Kapital auf ganze Mark abgerundet Das abgerundete Kapital ist in der Variablen **kapgerundet** abgelegt, dadurch ist das Kapital selbst weiterhin in der Variablen **kap** verfügbar.

```
INPUT "Kapital? ", kap
INPUT "Zinsfuß? ", p
kapgerundet=INT(kap)
zins=kapgerundet*p/100
PRINT "Jahreszins:";zins
```

81) Es soll außer dem Jahreszins auch das Kapital nach einem Jahr ausgegeben werden. Verbessere außerdem die Bildschirmgestaltung.

Eine selbstdefinierte Abrundefunktion

Dora ist immer noch nicht zufrieden: Bei 3 DM Kapital und einem Zinssatz von 2,75% errechnet der Computer Jahreszinsen von 0,0825 DM. Das Ergebnis sollte noch auf zwei Stellen hinter dem Dezimalpunkt abgerundet werden. Berta sucht in der Hilfe eine *Standardfunktion*, die das leistet, findet aber nichts.
Aber eine kurze Überlegung hilft weiter:
Den DM-Betrag auf 2 Stellen abrunden heißt, den Betrag in Pfennigen auf Ganze abzurunden. Der Betrag muss also erst mit 100 multipliziert werden, auf das Ergebnis wird die **INT**-Funktion angewandt und schließlich wird durch 100 dividiert. Die gesucht Funktion (Berta nennt sie **abger**) kann also mit Hilfe von **INT** definiert werden.

Das sieht dann so aus:

```
FUNCTION abger(x)
 abger = INT(100*x)/100
END FUNCTION
```

In der zweiten Zeile steht, wie der Funktionswert berechnet wird. Anders als in der Mathematik üblich, steht hierbei links vom Gleichheitszeichen nur der Funktionsname.

Füge diese Funktionsdefinition in das Programm ein. Sobald du die erste Zeile der obigen Funktionsdefinition eingegeben hast, macht Q-BASIC ein eigenes Fenster für diese Funktion auf. Um zum eigentlichen Programm, dem *Hauptprogramm*, zurückzukehren, wähle den Menüpunkt *Ansicht - SUBS* oder drücke die Taste F2. Du siehst dann, dass unser Programm jetzt aus zwei Teilen besteht, nämlich dem Hauptprogramm (wahrscheinlich noch mit dem Namen ´Unbenannt´) und der Funktionsdefinition. Du kannst auswählen, welchen Teil du auf dem Bildschirm betrachten und verändern willst.

In das Hauptprogramm wird eine Zeile eingefügt:

```
zins = abger(zins)
```

82) Füge diese Zeile und die Definition der Funktion **abger** dem Programm hinzu. Teste das Programm mit geeigneten Werten. Speichere das Programm ab. Danach hat sich das Hauptprogramm etwas verändert. Am Anfang des Hauptprogrammes hat Q-BASIC eine Zeile eingefügt, die mit dem Schlüsselwort DECLARE darauf hinweist, dass ein Funktion **abger** definiert ist.

Du kannst die Funktion **abger** auch im Direktmodus testen:

```
PRINT abger(1.234)
PRINT abger(1.999)
```

83) Berta kann jetzt die Entwicklung ihres Kapitals auf den Pfennig genau vorausberechnen:

```
'Kapitalentwicklung
INPUT "Kapital? ", kap
INPUT "Zinsfuß? ", p
INPUT "Anzahl der Jahre? ", n
PRINT
FOR i=1 TO n
  kapger=INT(kap)
  zins=kapger*p/100
  zins=abger(zins)
  kap=kap+zins
  PRINT USING "Nach ## Jahren: ####.## DM"; i, kap
NEXT i

FUNCTION abger(x)
  abger =  INT(100*x)/100
END FUNCTION
```

Immer wenn wir in einem Programm eine Zahl auf zwei Stellen hinter dem Dezimalpunkt abrunden müssen, werden wir uns an die Funktion **abger** erinnern und ihre Definition in das Programm aufnehmen.

84) Nach Eingabe des Nettopreises sollen Mehrwertsteuer und Bruttopreis ausgegeben werden.

85) Eine Firma gewährt ihren Kunden 3% Skonto, wenn sie innerhalb von 14 Tagen bezahlen. Schreibe ein Programm, das den Skontobetrag berechnet. Auch dieser DM-Betrag muss natürlich auf zwei Stellen hinter dem Dezimalpunkt abgerundet werden.

Funktionen aus der Mathematik

Funktionen sind dir in der Mathematik schon oft begegnet. Rechts steht ein Programm, das die Wertetafel einer Funktion f ausdruckt. Es ist $f(x) = 3x^2+4$. Du kannst dir im Direktmodus noch weitere Funktionswerte schreiben lassen:

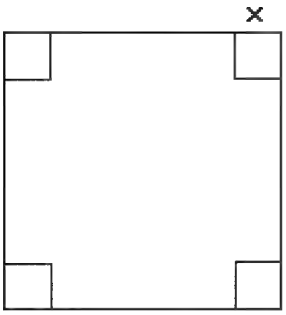
```
PRINT f(1.5) usw.
```

```
' Wertetafel
PRINT
PRINT "x","f(x)"
FOR x=-10 to 10
 PRINT x,f(x)
NEXT x

FUNCTION f(x)
 f = 3*x*x+4
END FUNCTION
```

86) Ändere das Programm so ab, dass eine Wertetafel der Funktion $f(x) = -4x^2+2x$ ausgegeben wird. Verbessere die Ausgabe durch Leerzeilen, **CLS** und **PRINT USING**.

87) Aus einem quadratischen Stück Papier (Breite 1 m) soll durch Herausschneiden von 4 Quadraten und anschließendes Falten eine (oben offene) Schachtel entstehen. Wie muss man die Länge x wählen, um das Volumen der Schachtel möglichst groß zu machen? Zunächst ermitteln wir, wie das Volumen V von x abhängt. Es ist



$$V(x) = x*(1-2x)^2 .$$

Diese Funktion kannst du in Q-BASIC definieren:

```
FUNCTION v(x)
  v = x*(1-2*x)^2
END FUNCTION
```

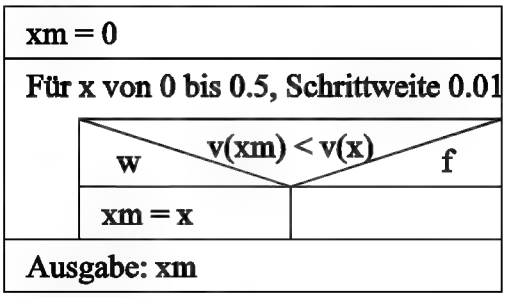
Anschließend kannst du im Direktmodus verschiedene x-Werte ausprobieren, z.B.

```
PRINT v(0.1)
```

So kann man sich an den Wert herantasten, der maximales Volumen ergibt (beachte hierbei den Definitionsbereich von x).

88) Suche den Maximalwerte des Volumens jetzt, indem du eine Tabelle der Funktionswerte ausdrucken lässt, und dann in dieser Tabelle den maximalen Wert suchst.

89) Das Programm soll jetzt selbst den x-Wert finden, bei dem das Volumen maximal wird. Im nebenstehenden Struktogramm ist der Algorithmus angegeben: Schritt für Schritt werden die x-Werte durchlaufen und der bisher beste Wert in xm gespeichert.



90) Berta möchte ein Programm schreiben, das ihr ausdruckt, wie lange es dauert, bis sich ihr Kapital verdoppelt hat. Sie möchte dies für verschiedene Zinssätze machen. Ihr Ziel ist eine Tabelle, wie sie nebenstehend angedeutet ist. In 42) ist ein Teil dieser Aufgabe schon erledigt, versuche darauf aufzubauen.

p	Zeit
1	70
2	36
3	24
4	18
5	15
6	12
7	11
8	10
9	9
10	8

91) Berta hat 90) leider nicht geschafft. Sie sucht mal wieder Hilfe bei Dora. Diese schreibt ihr eine Funktion, die aus dem Zinsfuß p die Verdoppelungszeit berechnet (siehe rechts). Erprobe diese Funktion im Direktmodus:

```
PRINT verdopzeit(3)
```

usw. Schreibe dann das verlangte Programm mit Hilfe dieser Funktion.

```
FUNCTION verdopzeit(p)
 kap=1
 zeit=0
 WHILE kap<2
  zeit=zeit+1
  zins=kap*p/100
  kap=kap+zins
 WEND
 verdopzeit = zeit
END FUNCTION
```

Das letzte Beispiel zeigt nochmal deutlich, wie Funktionen helfen, ein Programm übersichtlicher zu gestalten, zu *strukturieren.*

Unterprogramme (Prozeduren)

Prozeduren dienen genau wie Funktionen dazu, ein Programm zu *strukturieren.*

Clemens will ein Zahlenratespiel programmieren. Dazu schreibt er zunächst eine Gliederung:

Spielanleitung
Spiel
Verabschiedung

```
' Zahlenratespiel
anleitung
spiel
verabschiedung
END
```

Wie du siehst, hat Clemens die Gliederung gleich ins Programm übernommen, diese 4 Zeilen bilden sein *Hauptprogramm.* Jetzt muss er noch beschreiben, wie die *Unterprogramme* (Prozeduren) **anleitung**, **spiel** und **verabschiedung** aussehen. Das Schlüsselwort **SUB** ist eine Abkürzung des englischen Wortes *subroutine.*

92) Nebenstehend siehst du die Definitionen der Unterprogramme **anleitung** und **spiel**. Schreibe selbst das Unterprogramm **verabschiedung.**

```
SUB anleitung
  CLS
  PRINT "Guten Tag,"
  PRINT "Ich möchte mit dir spielen."
  PRINT "Ich denke mir eine Zahl"
  PRINT "zwischen 1 und 100,"
  PRINT "du musst sie raten."
END SUB
```

Versuche die Prozedur **spiel** zu verstehen. Rufe sie auch im Direktmodus auf. Spiele das Spiel mehrmals; was fällt dir dabei auf? Ergänze das Hauptprogramm am Anfang durch die Zeile

```
RANDOMIZE TIMER
```

Was bewirkt diese Zeile?

```
SUB spiel
 z = 1 + INT(100 * RND)
 DO
  INPUT "Rate! ", x
  IF x < z THEN
   PRINT "Deine Zahl ist zu klein."
  ELSEIF x > z THEN
   PRINT "Deine Zahl ist zu groß."
  ELSE
   PRINT "Richtig geraten."
  END IF
 LOOP UNTIL x=z
END SUB
```

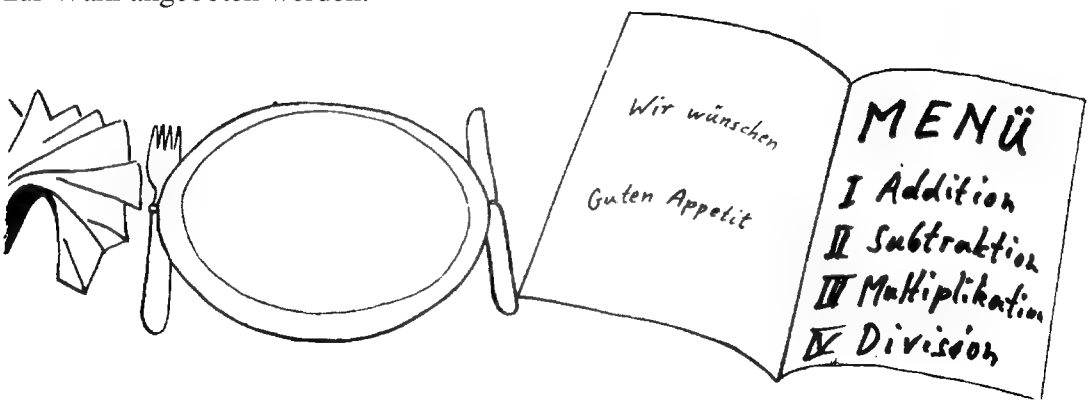
93) Manche Leute brauchen viele Rateversuche, andere wenige, das hängt teilweise vom Zufall ab. Wer allerdings mehr als 7 Versuche braucht, hat ungeschickt geraten. Wie lautet eine geschickte Strategie?

94) Das Programm soll die Anzahl der Versuche mitzählen und am Schluss dem Spieler mitteilen. Dazu richten wir in der Prozedur **spiel** eine Variable **versuchszahl** ein, die wir zunächst auf 0 setzen und dann bei jedem Durchlaufen der DO-LOOP-Schleife um 1 höher setzen. Sobald der Benutzer richtig geraten hat, wird ihm diese Zahl der Versuche mitgeteilt.

95) Das Programm soll die Anzahl der Rateversuche beurteilen und kommentieren. Genügte ein Versuch, so war das sicher unglaubliches Glück, bei 100 Versuchen kamen Pech und Ungeschicklichkeit zusammen. Schreibe eine Prozedur **bewertung**, die die Anzahl der Rateversuche mitteilt und bewertet (ohne beleidigend zu werden). Diese Prozedur soll nach **spiel** aufgerufen werden. Versuche das zu programmieren. Du stößt dabei auf ein Problem: Die Prozedur **bewertung** braucht den Inhalt der Variable **versuchszahl** aus der Prozedur **spiel**. Nun werden in Q-BASIC Variablen innerhalb einer Prozedur als *lokal* betrachtet, wenn nichts anderes festgelegt ist, das heißt, dass die Variable außerhalb der Prozedur unbekannt ist. Wenn man will, dass die Variable global sein soll, also in allen Teilen des Programms bekannt sein soll, muss man das ins Hauptprogramm mit Hilfe des Schlüsselwortes **SHARED** schreiben. In unserem Fall lautet der Befehl

```
DIM SHARED versuchszahl
```

96) Anton will für seinen Bruder Felix ein Trainingsprogramm für die 4 Grundrechenarten schreiben. Es soll ein *Menü-Programm* werden, das heißt, die vier Möglichkeiten sollen wie auf einer Speisekarte zur Wahl angeboten werden.



Unten siehst du das fertige Hauptprogramm. Es sind jetzt noch die Prozeduren **addition**, **subtraktion**, **multiplikation** und **division** zu schreiben. Rechts steht ein Vorschlag für **addition**. Schreibe die fehlenden Prozeduren.

```
' Rechentraining
 RANDOMIZE TIMER
 DO
   PRINT "    M E N Ü"
   PRINT
   PRINT "Addition....1"
   PRINT "Subtraktion.2"
   PRINT "Multiplikat.3"
   PRINT "Division....4"
   PRINT "Ende........5"
   PRINT
   INPUT "Wahl? ", wahl
   IF wahl=1 THEN
     addition
   ELSEIF wahl=2 THEN
     subtraktion
   ELSEIF wahl=3 THEN
     multiplikation
   ELSEIF wahl=4 THEN
     division
   END IF
 LOOP UNTIL wahl = 5
```

```
SUB addition
  CLS
  x=2+INT(98*RND)
  y=2+INT(98*RND)
  PRINT x;"+",y;"=";
  INPUT z
  PRINT
  IF z=x+y THEN
    PRINT "Richtig."
  ELSE
    PRINT "Falsch."
    PRINT "Erg.:";x+y
  END IF
  PRINT
END SUB
```

97) Strukturiere das obige Programm noch stärker durch Einführung von Prozeduren **menuezeigen** und **waehlen** und einer Funktion **zufallszahl(a,b)**.

98) Bei falscher Antwort soll dem Benutzer nochmals die Möglichkeit zum Rechnen gegeben werden, nach 2 falschen Antworten soll das richtige Ergebnis genannt werden.

99) Anton will in seinem Trainingsprogramm (Aufgabe 96) immer den Bildschirm löschen, bevor das Menü erscheint. Er ergänzt deshalb den Befehl CLS an entsprechender Stelle.
Die Wirkung ist fatal. Der Bildschirm wird so schnell gelöscht, dass das Ergebnis der vorherigen Rechnung nicht mehr gelesen werden kann. Anton übernimmt aus einem Buch die Prozedur **warteauftaste**.

```
SUB warteauftaste
  PRINT
  PRINT "Weiter: Drücke irgendeine Taste"
  SLEEP
END SUB
```

Diese Prozedur lässt das Programm scheinbar anhalten, bis irgendeine Taste gedrückt wird. Anton ruft diese Prozedur jeweils nach der Ausgabe des Ergebnisses auf. Auch wenn er diese Prozedur nicht ganz durchschaut, kann er sie doch ruhigen Gewissens als neuen Befehl verwenden.

100) Auf der Diskette befindet sich eine benutzerfreundlichere Fassung der Aufgabe 96. Dabei kann man mit den Richtungstasten zwischen den Menüpunkten auswählen. Dies bedeutet für das Programmieren allerdings mehr Aufwand. Selbst wenn dir die neuen Prozeduren **menuezeigen** und **waehlen** unverständlich sind, kannst du sie als Bausteine verwenden.

101) Beim Prozentrechnen treten die 3 Größen Grundwert, Prozentsatz, Prozentwert auf. Sind zwei davon bekannt, kann die dritte berechnet werden. Schreibe ein Menü-Programm, das den Benutzer zwischen den 3 Möglichkeiten auswählen lässt und nach Eingabe der zwei bekannten Größen die dritte berechnet.

2.5 Ergänzungen

Teilen

Wir betrachten nochmals das Eierpackproblem 46. Es war die Aufgabe, eine natürliche Zahl durch 6 zu teilen und den Rest zu ermitteln. Dies wurde mit Hilfe einer **WHILE**-Schleife auf wiederholte Subtraktion zurückgeführt. Für große Zahlen ist das natürlich unbefriedigend. In Q-BASIC gibt es für diesen Zweck die Operationen \ und **MOD** :

x \ y ist das Ergebnis der ganzzahligen Division von x durch y
x MOD y ist der dabei entstehende Rest.

102) Experimentiere im Direktmodus mit \ und **MOD** :

```
PRINT 23 \ 5                PRINT -7 \ 5
PRINT 23 mod 5              usw.
```

103) Schreibe Aufgabe 46 mit Hilfe von \ und **MOD** um.

104) Wir schreiben ein Programm, das alle Teiler einer natürlichen Zahl ausgibt.

```
'Teiler einer Zahl n
DIM n AS INTEGER, i AS INTEGER
INPUT "n? ", n
PRINT "Teiler von";n,":";
FOR i=1 TO n
  IF n MOD i=0 THEN PRINT i;
NEXT i
```

In der zweiten Zeile haben wir festgehalten, dass n und i Variablen für ganze Zahlen sein sollen. Das führt dazu, dass der Computer mit diesen Variablen einfacher und genauer rechnen kann.

Außerdem haben wir von der Möglichkeit Gebrauch gemacht, eine IF-Anweisung in *eine* Zeile zu schreiben, falls die zweite Möglichkeit entfällt.

105) Ist i ein Teiler von n, so ist auch n/i ein Teiler von n. Ändere das vorige Programm so ab, dass die Teiler immer paarweise geschrieben werden.

```
Teiler von 18            Teiler von 16

1     18                   1     16
2     9                    2     8
3     6                    4     4
6     3                    8     2
9     2                    16    1
18    1
```

106) Dies führt auf eine Idee, die viel Zeit spart. Jeder Teiler kommt jetzt in der Liste zweimal vor, es genügt also, die halbe Liste auszugeben. Wie man an den obigen Beispielen sieht, genügt es, die Teiler i zu betrachten, für die $i*i \leq n$ ist.
Wir ändern deshalb die **FOR**-Schleife in eine **WHILE**-Schleife um.

```
'Teiler einer Zahl n
DIM n AS INTEGER, i AS INTEGER
INPUT "n? ", n
PRINT "Teiler von";n;":"
i=1
WHILE i*i<=n
  IF n MOD i=0 THEN PRINT i,n \ i
  i=i+1
WEND
```

Überlege dir, wie groß die Zeiterparnis etwa bei der Zahl 101 ist. Die größere Schnelligkeit ist mit zwei Nachteilen erkauft:

- die Teiler sind nicht mehr der Größe nach geordnet,
- bei Quadratzahlen erscheint ein Teiler doppelt (leicht zu beheben).

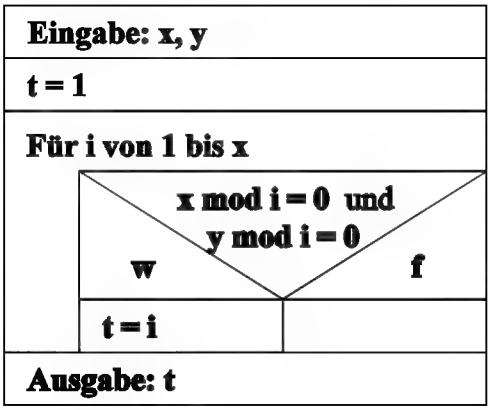
107) Die Zahl 6 ist die Summe ihrer echten Teiler:

$$6 = 1 + 2 + 3$$

Gibt es noch weitere Zahlen mit dieser Eigenschaft? Um dies zu entscheiden, schreibe zunächst eine Funktion **teilersu**(n), die die Teilersumme liefert. Damit kannst du dann leicht eine Tabelle erzeugen und aus dieser die Zahlen mit der gewünschten Eigenschaft herauslesen.

108) Bequemer ist es, den Computer suchen zu lassen. Er soll jetzt nur noch die gesuchten Zahlen ausdrucken.

109) Wir wollen den größten gemeinsamen Teiler zweier Zahlen x und y bestimmen. Eine nahe liegende Lösung ist im Struktogramm angegeben. Der Variablen t wird zunächst der kleinste Teiler (nämlich 1) zugewiesen, dann geht man in die Schleife und sucht größere gemeinsame Teiler. Beim Verlassen der Schleife ist in t der größte gemeinsame Teiler gespeichert. Wie du im Struktogramm siehst, muss man zwei Bedingungen mit **und** verknüpfen. Das entsprechende Wort in Q-BASIC heißt **AND**.



110) Der ggT von 2 und 1000000 ist schnell gefunden; zum Bestimmen des ggT von 1000000 und 2 wird viel Zeit benötigt. Das kann man abstellen, wenn man die Zählschleife nicht bis x, sondern bis **minimum**(x,y) laufen lässt. Dabei ist das Minimum definiert durch

```
FUNCTION minimum(x,y)
  IF x<=y THEN
    minimum =  x
  ELSE
    minimum =  y
  END IF
END FUNCTION
```

Primzahlen

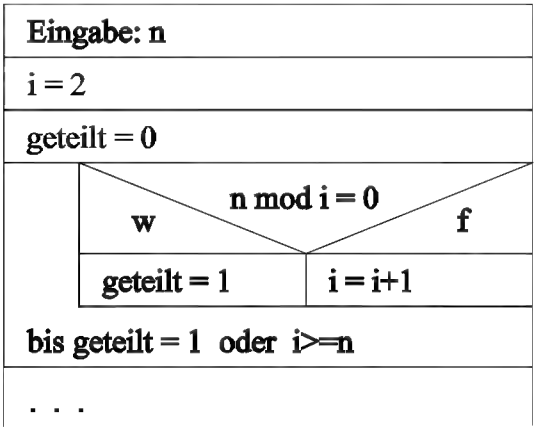
111) Eine Primzahl ist eine Zahl, die größer als 1 ist und genau zwei Teiler hat, nämlich 1 und sich selbst. Peter schreibt ein Programm zum Erkennen von Primzahlen.

```
'Peters Primzahltest
DIM n AS INTEGER, i AS INTEGER
INPUT "n? ", n
geteilt=0
FOR i=2 TO n-1
  IF n MOD i=0 THEN geteilt=1
NEXT i
IF geteilt=1 THEN
  PRINT n;"ist keine Primzahl."
ELSE
  PRINT n;"ist eine Primzahl."
END IF
```

Die Variable **geteilt** ist eine so genannte *Flagge*, welche die Werte **1** oder **0** annehmen kann. In dieser Variablen wird notiert, ob n schon geteilt worden ist.

112) Bei großen Zahlen braucht das Programm viel Zeit. Paul meint, das sei oft unnötig. Für die Zahl 1000000 steht ja sofort fest, dass 2 ein Teiler ist; es ist unsinnig, jetzt noch weiterzusuchen. Paul überlegt sich eine Lösung im Struktogramm. Die Schleife wird nur durchlaufen, bis ein Teiler gefunden ist **oder** alle möglichen Teiler probiert wurden. In Q-BASIC lautet diese Bedingung

```
UNTIL geteilt=1 OR i>=n
```



113) Pauls Programm erkennt die Zahl 2 nicht als Primzahl. Um diesen Fehler zu beheben, musst du die **UNTIL**-Schleife durch eine **WHILE**-Schleife ersetzen. Außerdem betrifft Pauls Verbesserung nur Zahlen, die teilbar sind. Bei Primzahlen bringt die Überlegung aus Übung 106 großen Zeitgewinn: Es genügt nach Teilern zu suchen, solange i*i <= n ist. Gibt es keinen Teiler, der dieser Bedingung genügt, so gibt es überhaupt keinen. Die Schleife soll also durchlaufen werden,

solange geteilt = 0 **und** i*i<=n ist.

Die Übersetzung in Q-BASIC lautet:

```
WHILE geteilt = 0 AND i*i<=n
```

114) Ist 2 kein Teiler der Zahl n, so ist es sinnlos, nach weiteren geraden Teilern zu suchen. Wir verändern den Algorithmus entsprechend:
Der Fall i=2 wird am Anfang gesondert behandelt. Falls 2 kein Teiler ist, treten wir mit i=3 in die Schleife ein und setzen i ab jetzt immer um 2 höher, sodass nur noch die ungeraden Zahlen drankommen.

```
'Primzahltest
DIM n AS INTEGER, i AS INTEGER
INPUT "n? ", n
geteilt= 0
IF n MOD 2=0 THEN geteilt=1
i=3
WHILE geteilt=0 AND i*i<=n
  IF n MOD i=0 THEN geteilt=1
  i=i+2
WEND
IF geteilt=1 THEN
  PRINT n;"ist nicht prim."
ELSE
  PRINT n;"ist prim."
END IF
```

Auch dieses Programm scheitert an der Zahl 2. Verbessere diesen Fehler.

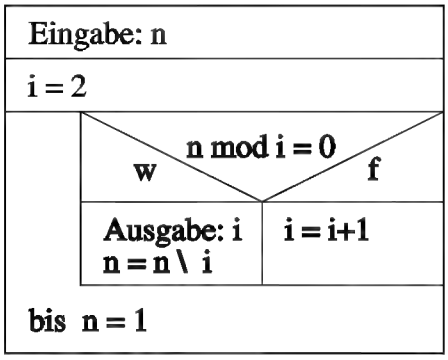
115) Nach Eingabe einer Zahl g sollen alle Primzahlen, die kleiner oder gleich g sind, aufgelistet werden.

116) Nach Eingabe von u und g sollen alle Primzahlen zwischen u und g aufgelistet werden.

117) Schreibe ein Programm, das eine Zahl in ihre Primfaktoren zerlegt. Der Dialog soll etwa so aussehen:

```
n? 90
Primfaktoren:  2  3  3  5
```

Der Algorithmus ist im Struktogramm angegeben: n wird solange durch 2 geteilt, wie dies geht, dann durch 3 usw. Ist i nicht prim, so kann die Division nicht aufgehen, da n schon durch die Teiler von i dividiert wurde. Also werden nur Primfaktoren ausgedruckt.



118) Man gewinnt auch hier Zeit, wenn man außer der 2 nur ungerade Teiler probiert.

119) Bestimme den ggT zweier Zahlen n und m als das Produkt seiner Primfaktoren. Der Algorithmus lehnt sich eng an Übung 117 an, statt n MOD i = 0 wird jetzt geprüft

```
n MOD i = 0  AND  m MOD i = 0.
```

Bruchrechnen

120) Berta hat Schwierigkeiten beim Bruchrechnen und wünscht sich ein Programm, das ihr die Grundrechenarten durchführt. Folgender Dialog schwebt ihr vor:

```
M E N Ü  Bruchrechnen:

Addition........1
Subtraktion.....2
Multiplikation..3
Division........4
Ende............5

Wahl?  1
```

```
Zähler der 1. Zahl?  2
Nenner der 1. Zahl?  3
Zähler der 2. Zahl?  5
Nenner der 2. Zahl?  7

   2/3 + 5/7 = 29/21
```

Das Hauptprogramm kannst du vom Rechentraining (96) übernehmen. Nur die Prozeduren **addition**, **subtraktion**, **multiplikation** und **division** sehen jetzt natürlich anders aus. Wir geben dir hier die Prozedur **addition** an, die anderen kannst du sicher selbst schreiben.

```
SUB addition
  eingabe
  nenner=n1*n2
  zaehler=z1*n2+z2*n1
  PRINT z1,"/",n1"+";z2,"/",n2;"=";zaehler,"/",nenner
END SUB
```

Schön wäre es, wenn das Ergebnis in gekürzter Form vorläge. Schreibe dazu mit Hilfe von Übung 109 oder 119 eine Funktion ggt(x,y) und füge in das Programm ein:

```
g=ggt(zaehler, nenner)
zaehler= zaehler \ g
nenner= nenner \ g
```

121) Verbessere die Ausgabe mit **CLS** und der Prozedur **warteauftaste** aus Aufgabe 99.

122) Schreibe ein Trainingsprogramm fürs Bruchrechnen. Der Computer stellt eine Aufgabe, der Benutzer gibt sein Ergebnis ein und erfährt, ob es richtig ist (Vorsicht: Vielleicht hat er nur das Kürzen vergessen).

Stellenwertsysteme

123) Wir wollen die Quersumme einer Zahl n berechnen, z.B. n = 13245.
Wir können uns die Ziffern auf elegante Weise verschaffen, indem wir hinten beginnen.
n **MOD** 10 ergibt die Einerziffer 5,
n \ 10 ergibt 1324, also die Zahl, die beim Wegstreichen der Einerziffer übrig bleibt.
So können wir weitermachen:
1324 **MOD** 10 ergibt die Ziffer 4 usw. Das wiederholen wir, bis alle Ziffern weggestrichen sind.
Wenn wir die Dezimalziffern alle addieren, erhalten wir die Quersumme.

```
'Quersumme
DIM n AS INTEGER, x AS INTEGER, quersu AS INTEGER
INPUT "Wie heißt deine Zahl?", n
x=n
quersu=0
WHILE x>0
  ziffer = x MOD 10
  quersu = quersu+ziffer
  x = x \ 10
WEND
PRINT "Die Quersumme von";n;"ist";quersu
```

124) Im Dualsystem wird als Basis die Zahl 2 benützt. Es gibt nur die Ziffern 0 und 1, z. B. bedeutet

$$10111 = 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0$$

Die Dualziffern können ähnlich wie die Dezimalziffern berechnet werden, die Einerziffer ist jetzt n **MOD** 2. Wir wollen nach Eingabe der Dezimaldarstellung die Dualdarstellung ausrechnen lassen. Dabei gibt es noch eine Schwierigkeit: Wir arbeiten die Zahl n von hinten ab; wenn wir die Dualziffern nacheinander schreiben lassen, stehen sie in verkehrter Reihenfolge. Wir helfen uns zunächst damit, dass wir die Dualziffern in einer Tabelle festhalten.
Schreibe ein Programm, das die folgende Ausgabe bewirkt.

```
Dualdarstellung von 23:

       1: 1
       2: 1
       4: 1
       8: 0
      16: 1
```

125) Versuche einen Algorithmus zu entwickeln, der die Zahl n von vorne abarbeitet, sodass folgender Dialog entsteht:

```
n? 23
23 im Dualsystem: 10111
```

Hinweis: Zunächst muss festgestellt werden, welches die höchste Zweierpotenz ist, die in der Zahl enthalten ist.

Rechentraining

126) Je mehr wir mit dem Taschenrechner und Computer arbeiten, desto wichtiger wird unsere Fähigkeit, durch Überschlagsrechnung das Ergebnis zu überprüfen. Hier ist die einfache Version eines Trainingsprogramms zur überschlägigen Multiplikation.

```
'Überschlag
x= INT(1000*RND+1)
y= INT(1000*RND+1))
z=x*y
PRINT "Überschlage";x,"*",y
INPUT "Dein Ergebnis? ", erg
IF erg/z<1.25 AND erg/z>.75 THEN
  PRINT "Gut"
ELSE
  PRINT "Schlecht"
END IF
```

127) Das Programm hat noch Mängel:

Die Bewertung des Ergebnisses ist zu grob, es sollte zwischen "gut", "zu grob" und "falsch" unterschieden werden.
Um nicht immer wieder starten zu müssen, wird man das Programm in eine Schleife einbauen.
Schließlich könnte man zu genaue Ergebnisse, die erkennbar nicht durch Überschlag gewonnen wurden, zurückweisen lassen.

128) Eine häufige Aufgabe in Intelligenztests ist das Weiterführen von Zahlenfolgen:
Führe fort:

1, 3, 5, 7, . . .
4, 7, 10, 13, . . .
23, 19, 15, 11, . . .

Der Computer soll solche einfachen Zahlenfolgen erzeugen und die Antwort des Benutzers bewerten. Die obigen Folgen sind alle von der Bauart

$x(i) = a + b*i$

wobei a, b zufällige, feste Zahlen sind, und i die Werte 1,2,3, . . . durchläuft.

129) Da die obigen Folgen alle vom selben Typ sind, wird das Training bald langweilig. Wir wollen, dass auch Folgen der Form

$x(i) = a*i + b*i*i$
$x(i) = a*b^i$

in zufälliger Reihenfolge erscheinen. Pass aber auf, dass die Zahlen nicht zu groß werden.

Wurzeln

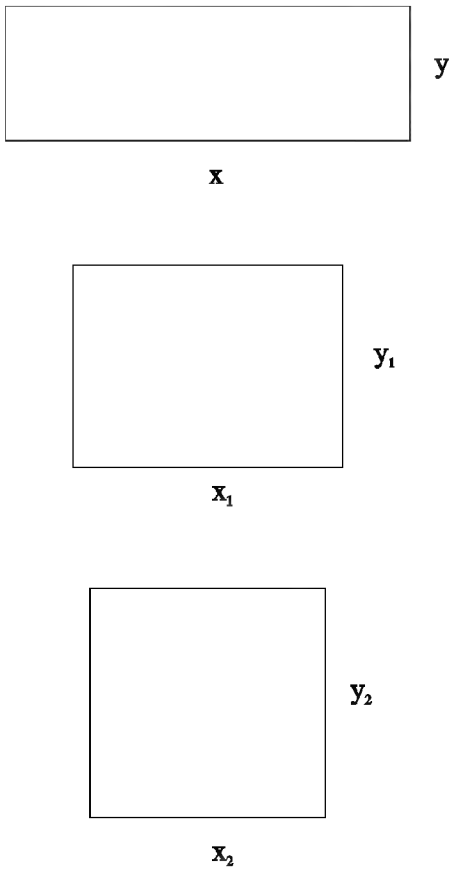
130) Wir betrachten ein Rechteck mit den Seiten $x = 6$ und $y = 2$ und suchen ein Quadrat, das denselben Flächeninhalt $a = 12$ hat. Da die gesuchte Quadratseite zwischen x und y liegen muss, liegt es nahe, das arithmetische Mittel

$x_1 = (x+y)/2$

zu probieren. Dies ergibt $x_1 = 4$, ist also nicht die gesuchte Quadratseite. Berechnen wir aber $y_1 = a/x_1$, so haben wir ein Rechteck mit den Seiten x_1, y_1 gefunden, das einem Quadrat schon näher kommt und denselben Flächeninhalt wie das ursprüngliche hat.
Auf das neue Rechteck wenden wir das Verfahren nochmals an, berechnen also

$x_2 = (x_1+y_1)/2$
$y_2 = a/x_2$

Dies ergibt zwar immer noch kein Quadrat, da x_2 und y_2 ungleich sind, aber der Unterschied der Seiten ist so gering, dass wir ihn in der Zeichnung kaum noch erkennen können.



Wir haben die gesuchte Quadratseite nicht gefunden, dafür aber einen Näherungswert, der für die Zeichnung genügt. Wollen wir einen besseren Näherungswert, so müssen wir das Verfahren noch einige Male anwenden. Schreibe nun ein Programm, das nach Eingabe der Rechteckseiten 10 mal das Verfahren anwendet und die neuen Rechteckseiten ausgibt. Die entscheidenden Zeilen lauten

```
x = (x+y)/2
y = a/x
```

131) Beim Testen des Programms mit verschiedenen Eingabewerten merkst du, dass zehnmaliges Wiederholen oft unnötig, manchmal aber nicht ausreichend ist. Ersetze die **FOR**-Schleife durch eine **WHILE**-Schleife. Breche die Wiederholung ab, wenn der Unterschied der beiden Rechteckseiten kleiner ist, als die Genauigkeit, mit der die Zahlen auf dem Bildschirm erscheinen.

132) Gegeben sei nun der Flächeninhalt a eines Quadrats, gesucht die Quadratseite x. Die Zahl x heißt dann die *Quadratwurzel* von a. Wandle das vorige Programm so ab, dass nach Eingabe von a ein *guter Näherungswert* für die Wurzel ausgeben wird. Schreibe eine Funktion **quadratwurzel**, sodass **quadratwurzel(r)** die Quadratwurzel von r ist (näherungsweise). Vergleiche mit der Standardfunktion **sqr** (sprich: squareroot).

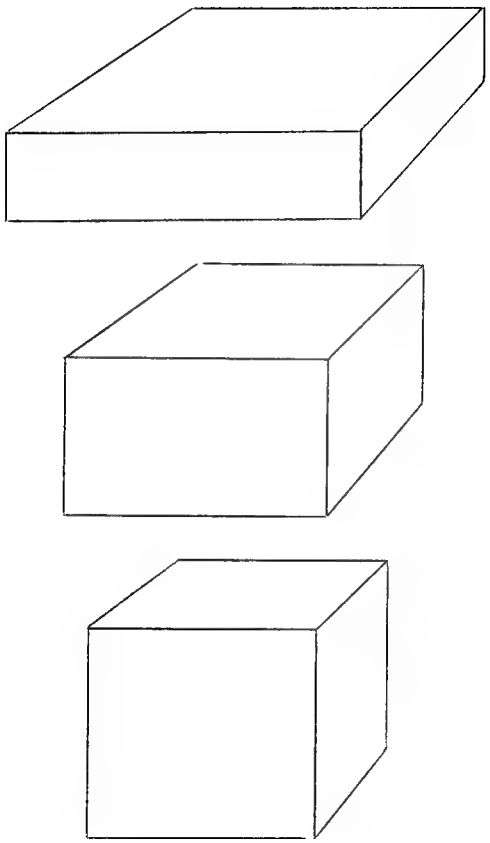
133) Ein ähnliches Problem im Raum lautet: Gegeben ist das Volumen v eines Würfels, gesucht seine Kantenlänge. Um zumindest einen Näherungswert zu finden, denken wir uns zunächst eine quadratische Säule mit der Grundkante x und der Höhe h, die das Volumen v hat (z.B. x=1, h=v), und wandeln diese dann in eine quadratische Säule um, die einem Würfel schon näher kommt. Der gesuchte Wert x muss zwischen den gegebenen drei Kantenlängen liegen, wir wählen daher für die Näherung x_1 das arithmetische Mittel

$x_1 = (x+x+h)/3$

Die Höhe h_1 wird so gewählt, dass sich das gegebene Volumen ergibt

$h_1 = v/(x_1 * x_1)$

Schreibe das entsprechende Programm.



134) Gilt $x^3 = v$, so heißt x die Kubikwurzel von v. Schreibe eine Funktion **kubikwurzel**.

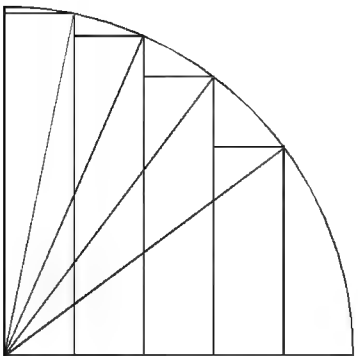
Die Zahl π

Hat ein Kreis den Radius r, so gilt für seine Fläche A und seinen Umfang U:
$A = \pi r^2$; $U = 2 \pi r$. Der Wert der Zahl π ist ungefähr 3,14. Wir wollen einen besseren Näherungswert berechnen. Dazu betrachten wir einen Viertelkreis mit Radius 1. Diesen Viertelkreis schöpfen wir mit Rechtecken der Breite 1/n aus, wobei n eine natürliche Zahl ist (siehe Skizze). Je größer n ist, desto genauer wird die Fläche bestimmt, multipliziert man ihren Wert mit 4, so erhält man einen Näherungswert für π. Im Fall n = 5 sind noch Radien eingezeichnet, die uns die Berechnung der Rechteckshöhen erleichtern. Nach Pythagoras gilt

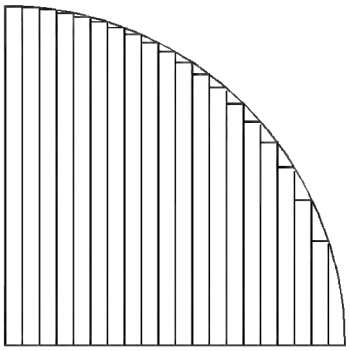
$$h_1 = \sqrt{1-(\frac{1}{5})^2}$$

$$h_2 = \sqrt{1-(\frac{2}{5})^2} \quad \text{usw.}$$

n = 5



n = 20



Also ist die Gesamtfläche aller Rechtecke:

$$A = \frac{1}{5} \cdot \left(\sqrt{1-\left(\frac{1}{5}\right)^2} + \sqrt{1-\left(\frac{2}{5}\right)^2} + \sqrt{1-\left(\frac{3}{5}\right)^2} + \sqrt{1-\left(\frac{4}{5}\right)^2} \right)$$

135) Übertrage diese Formel für eine beliebige natürliche Zahl n und schreibe darauf aufbauend ein Programm, das nach Eingabe von n die zugehörige Näherung von π ausgibt.

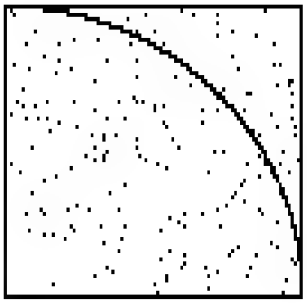
136) Es soll die nebenstehende Tabelle erzeugt werden:

n:	Näherung für PI
10	2.904518
100	3.120417
1000	3.141391

usw.

137) Die bisherigen Näherungswerte waren zu klein. *Überdeckt* man den Viertelkreis mit Rechtecken, so erhält man *zu große* Näherungswerte. Damit kann man π zwischen einem zu kleinen und einem zu großen Näherungswert einschließen und eine entsprechende Tabelle erzeugen.

138) Zum Schluss noch eine originelle, wenn auch ungenaue Methode zur Bestimmung von π. Auf das nebengezeichnete Quadrat mit Seitenlänge 1, in das ein Viertelkreis eingezeichnet ist, lassen wir in Gedanken zufällig Regentropfen fallen und zählen, wie viele davon in den Viertelkreis fallen.



Die Anzahl der Tropfen im Viertelkreis verhält sich zur Gesamtanzahl ungefähr wie die entsprechenden Flächen also wie $\pi/4 : 1$.

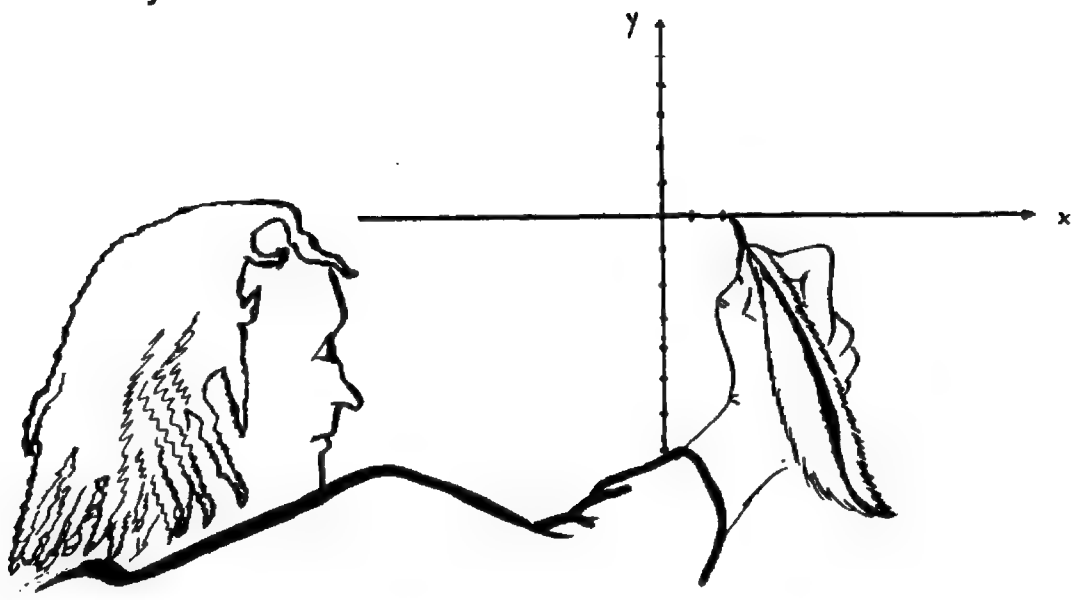
Einen zufälligen Regentropfen simulieren wir, indem wir die Koordinaten x und y jeweils zufällig zwischen 0 und 1 wählen lassen.

```
'Bestimmung von PI durch Regentropfen
RANDOMIZE TIMER
INPUT "Wie viele Regentropfen? ",n
anzahl=0
FOR i=1 TO n
  x=RND
  y=RND
  IF x*x+y*y <=1 THEN anzahl=anzahl+1
NEXT i
verhaeltnis=anzahl/n
naeh=4*verhaeltnis
PRINT "Näherung für PI:"; naeh
```

Es ist schön, wenn man das Fallen der Regentropfen auch auf dem Bildschirm verfolgen kann. Das entsprechende Programm "Regentropfen" ist im Anhang abgedruckt; die dafür notwendigen Befehle wirst du in Kapitel 3 kennen lernen.

3 Koordinatengrafik

3.1 Koordinatensysteme



Es ist umstritten, wer als erster auf den Gedanken kam, den Verlauf einer Funktion mit Hilfe eines Koordinatensystems darzustellen. Sicher ist nur, dass diese Idee sich durchsetzte, und so erwarten wir auch von unserem Computer, dass er mit Koordinaten umgehen kann.

Bildschirmkoordinaten, Punkte und Strecken

Um den Grafikbildschirm von Q-BASIC kennen zu lernen, geben wir zunächst ein kleines Programm ein:

```
SCREEN 12
LINE (0,0) - (300,200)
```

Durch den ersten Befehl werden die Grafikmöglichkeiten von Q-BASIC aktiviert. Die Zahl 12 bedeutet dabei unter anderem, dass die VGA-Grafik vorausgesetzt wird. Wenn dein Computer eine andere Grafikkarte hat, musst du statt dessen eine andere Zahl eingeben, etwa

```
SCREEN 10
```

Näheres erfährst du bei der Hilfe unter den Stichworten SCREEN-Anweisung, Bildschirmmodi. Du kannst aber auch ausprobieren, welche Zahl bei deinem Computer die günstigsten Ergebnisse liefert. Der zweite Befehl bewirkt das Zeichnen einer Strecke von (0,0) bis (300,200).

1) Ersetze die Zahl 12 durch andere Zahlen und schaue, ob sich das Bild verbessert. Ersetze die Zahlen 300 und 200 durch andere Zahlen, und schaue, wie sich die Linie auf dem Bildschirm verändert. Ersetze auch (0,0) durch einen anderen Punkt.

Du siehst: Bei (300,200) gibt die erste Zahl die x-Koordinate, die zweite Zahl die y-Koordinate an, wie wir es auch aus der Mathematik gewöhnt sind. Das Koordinatensystem beginnt allerdings in der linken oberen Ecke mit dem Punkt (0,0), geht nach rechts bis zum x-Wert 639, nach unten bis zum y-Wert 479 (siehe Skizze).

Diese Werte hängen allerdings davon ab, welchen Grafikmodus du gewählt hast, d.h. welche Zahl du hinter SCREEN angegeben hast.

Der Bildschirm besteht aus vielen kleinen Bildpunkten, so genannten Pixels (picture elements). Das Pixel an der Stelle (100,50) können wir in weiß darstellen durch den Befehl

```
PSET (100,50)
```

2) Gib diesen Befehl im Direktmodus ein und suche das weiße Pixel auf dem Bildschirm. Zeichne weitere Punkte und erkunde nochmals die Grenzen des Bildschirms. Wo ist der Punkt (639,0)? Wird der Punkt (640, 100) noch eingezeichnet. oder ist er schon außerhalb des Bildschirms? Gib auch PSET-Befehle in das Programm ein und starte neu.

Vielleicht willst du jetzt wissen, wie man den Bildschirm wieder sauber macht. Das geht mit dem bekannten Befehl

```
CLS
```

Alles klar? Probiere es im Direktmodus aus. Dann kannst du noch ein bisschen weiterzeichnen.

Sicher hat es dich gestört, dass die y-Koordinate von oben nach unten gezählt wird. Aus der Mathematik sind wir das anders gewöhnt. Um den Bildschirm an unsere Gewohnheiten anzupassen, gib jetzt als neues Programm ein

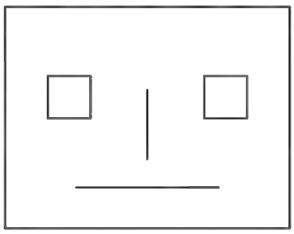
```
SCREEN 12
WINDOW (0,0)-(639,479)
```

Mit der WINDOW-Anweisung geben wir an, welche Koordinaten die linke untere Ecke haben soll, hier (0,0), und welche Koordinaten die rechte obere Ecke haben soll, hier (639,479). Wenn du in einem anderen Bildschirmmodus arbeitest, solltest du für die rechte obere Ecke die entsprechenden Koordinaten wählen.

3) Gib dieses Programm ein. Zeichne zunächst im Direktmodus einige Punkte, um dich auf dem Bildschirm zurechtzufinden. Zeichne dann das Quadrat mit den Eckpunkten (100,100), (100,200), (200,200), (200,100). Zeichne auch den Mittelpunkt des Quadrats. Wähle andere Koordinaten, sodass das Quadrat in der Mitte des Bildschirms sitzt.

Wenn du im Direktmodus zeichnest, kannst du zwar sofort sehen, was deine Anweisungen bewirken, aber sobald du einen Fehler machst, musst du von vorn beginnen. Deshalb ist es bequemer, eine Zeichnung als Programm zu schreiben. Du kannst dann verbessern, verändern und speichern.

4) Schreibe ein Programm zum Zeichnen des nebenstehenden Robotergesichtes.



Falls du einen Farbmonitor hast, kannst du das Bild auch in verschiedenen Farben zeichnen. In Q-BASIC kannst du 16 verschiedene Farben einsetzen, die durch die Nummern 0 bis 15 beschrieben werden. Diese Nummer muss hinter dem PSET- oder LINE-Befehl angegeben werden. Zum Beispiel zeichnest du einen roten Punkt an die Stelle (100,100) mit

```
PSET (100,100),12
```

und eine grüne Strecke von (10,100) nach (600,100) mit

```
LINE (10,100)-(600,100),10
```

5) Zeichne im Direktmodus oder mit Hilfe eines Programms 16 Linien in 16 verschiedenen Farben (Vorsicht: Eine der 16 Farben ist die Hintergrundfarbe, die Linie ist also nicht sichtbar). Was geschieht, wenn du eine Farbnummer angibst, die größer als 15 ist? Zeichne die Linien des Robotergesichts farbig.

Das Robotergesicht besteht im Wesentlichen aus Rechtecken. Da trifft es sich gut, dass der LINE-Befehl mit dem Zusatz B (wie Box) auch zum Zeichnen von Rechtecken benützt werden kann. Schau dir an, wie das Robotergesicht mit diesem Befehl gezeichnet werden kann. Vor dem Buchstaben B stehen zwei Kommas. Willst du das Rechteck farbig zeichnen, so muss zwischen diesen beiden Kommas die Nummer der Farbe stehen. Das Rechteck wird angegeben durch die Koordinaten der linken unteren Ecke und die Koordinaten der rechten oberen Ecke.

```
'Robotergesicht
SCREEN 12
WINDOW (0, 0) - (639, 479)
LINE (100, 100) - (220, 200),,B
LINE (120, 150) - (140, 170),,B
LINE (180, 150) - (200, 170),,B
LINE (140, 120) - (180, 120)
LINE (160, 130) - (160, 160)
```

6) Zeichne noch Ohren und Antennen zum Gesicht. Zeichne nach Möglichkeit farbig. Gibst du am Ende des LINE-Befehls statt des Buchstabens ´B´ ein ´BF´ ein, so wird das Rechteck in der jeweiligen Farbe ausgefüllt (Box Full). Probiere das beim Robotergesicht aus.

Mit Hilfe der Befehle PSET und LINE kannst du schon viele schöne Zeichnungen auf den Bildschirm zeichnen. Hier sind einige Vorschläge.

7) Versuche vorauszusagen, was die Programme leisten, bevor du sie laufen lässt. Ändere dann die Zahlen ab, um das Bild nach deinen Vorstellungen zu gestalten.

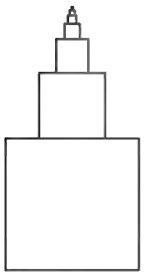
```
'Viele Linien
SCREEN 12
WINDOW (0, 0)-(639, 479)
FOR i=0 TO 400 STEP 50
 LINE (0, i) - (400-i,0)
NEXT i
```

```
'Viele Linien 2
SCREEN 12
WINDOW (0, 0)-(639, 479)
FOR i = 0 TO 400 STEP 50
 FOR k = 0 TO 400 STEP 50
   LINE (0, i)-(400, k)
  NEXT k
NEXT i
```

```
'Viele Rechtecke
SCREEN 12
WINDOW (0, 0)-(639, 479)
FOR i =15 TO 0 STEP -1
 LINE (0,0)-(10*i,10*i),i,BF
NEXT i
```

```
'Viele Rechtecke 2
SCREEN 12
WINDOW (0, 0)-(639, 479)
FOR i =0 TO 200 STEP 20
 LINE (1.5*i,i)-(2*i,2*i),,B
NEXT i
```

8) Auf ein Quadrat wird das nächste mit halber Seitenlänge gesetzt, auf dieses wieder eines mit halber Seitenlänge usw. (siehe rechts). Höre auf zu zeichnen, wenn die Seitenlänge kleiner als 1 Pixel wird. Arbeite mit einer WHILE-Schleife.

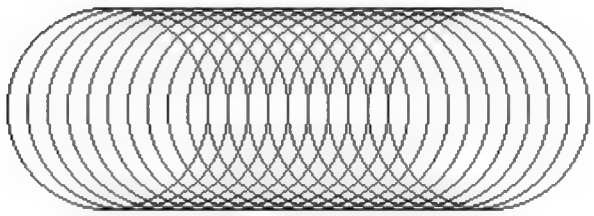


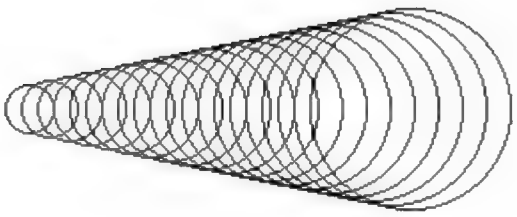
Kreise

Um einen Kreis zeichnen zu können, müssen wir wissen, wo sein Mittelpunkt liegen soll und wie groß sein Radius sein soll. Wollen wir z.B. einen Kreis um den Punkt (300,200) mit dem Radius 150 zeichnen, so lautet der Befehl

```
CIRCLE (300, 200), 150
```

9) Probiere diesen Befehl aus. Vergiss nicht, vorher die SCREEN-Anweisung zu geben. Versuche, eines der folgenden Bilder zu erzeugen. Verwende dazu eine FOR-Schleife.





10) Versuche, einige schöne Bilder mit Kreisen zu entwerfen.

11) Zeichne mit Hilfe von Kreisen und Strecken ein Clowngesicht.

Übergang zu anderen Koordinatensystemen

Das Koordinatensystem, in dem wir bis jetzt gezeichnet haben, ist für viele Fälle ungeeignet. Wollen wir ein anderes auf dem Bildschirm darstellen, das z.B. bei den x-Werten von -5 bis 6, bei den y-Werten von -3 bis 4 gehen soll, so geben wir ein

```
WINDOW(-5,-3)-(6,4)
```

Gib diesen Befehl ein und zeichne einige Punkte:

```
PSET (0,0); PSET (1,2); PSET (2.5,-2)
```

Wir zeichnen ein Quadrat und einen Kreis:

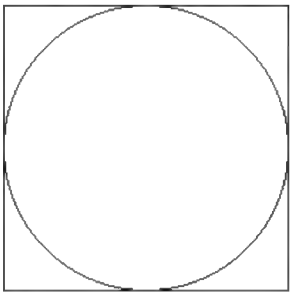
```
LINE (-2,-2) - (2,2),,B
CIRCLE (0,0), 2
```

Das Ergebnis entspricht nicht unseren Erwartungen. Das Quadrat wird nicht als Quadrat dargestellt; der Kreis sieht zwar wie ein Kreis aus, ist aber nicht dem Quadrat einbeschrieben. Um diesem Missstand abzuhelfen, müssen wir die Koordinaten in der WINDOW-Anweisung so wählen, dass das Verhältnis des y-Bereichs zum x-Bereich das Gleiche ist wie bei den Bildschirmkoordinaten, also wie 480 zu 640, d.h. wie 3 zu 4. Dies gelingt zum Beispiel mit

```
WINDOW(-5,-3.5)-(5,4)
```

Mit diesen neuen Zahlen sollte das Programm jetzt das Gewünschte leisten.

```
SCREEN 12
WINDOW (-5,-3.5)-(5,4)
LINE (-2,-2)-(2,2),,B
CIRCLE (0,0),2
```



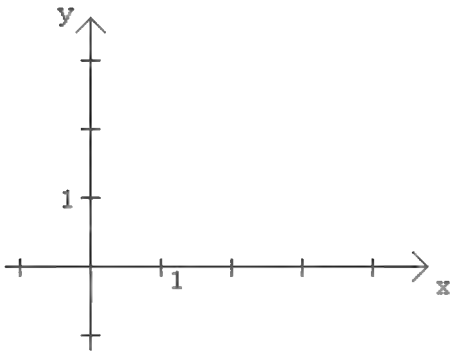
Falls du einen anderen Bildschirmmodus verwendest, musst du die Zahlen entsprechend abändern.

Damit man sich im Koordinatensystem zurechtfindet, zeichnet man meistens die Achsen ein. Auch wir wollen das jetzt tun.

Die x-Achse: `LINE (-5, 0) - (5, 0)`
Die y-Achse: `LINE (0, -3.5) - (0, 4)`

Als Nächstes wollen wir Teilungsstriche an den Achsen anbringen. Dies wäre im Direktmodus doch zu umständlich. Wir schreiben gleich Prozeduren, damit unsere Mühen sich auch später auszahlen. Dann sollten wir aber auch von den konkreten Zahlen -5, -3.5, 5, 4 des Beispiels loskommen. Wir schreiben deshalb ein Programm, das zunächst vom Benutzer die minimalen und maximalen x- bzw. y-Werte erfragt und dann das zugehörige Koordinatenkreuz zeichnet.

```
' Koordinatensystem
INPUT "xmin? ",xmin
INPUT "ymin? ",ymin
INPUT "xmax? ",xmax
INPUT "ymax? ",ymax
SCREEN 12
WINDOW (xmin,ymin)-(xmax,ymax)
achsenkreuz xmin,ymin,xmax,ymax
gitter xmin,ymin,xmax,ymax
```



Im Hauptprogramm steht schon, was wir wollen: Es soll das Achsenkreuz gezeichnet werden, wobei wir die Achsen mit Skalenstrichen versehen wollen. Zur besseren Orientierung wollen wir auch noch die Gitterpunkte mit ganzzahligen Koordinaten einzeichnen.

Die oben genannten Prozeduren sehen folgendermaßen aus:

```
SUB achsenkreuz (x1,y1,x2,y2)
  LINE (x1,0)-(x2,0)
  LINE (0,y1)-(0,y2)
  lx= 5*(x2-x1)/640
  ly= 5*(y2-y1)/480
  FOR x= INT(x1) TO INT(x2-0.4) : LINE(x,-ly)-(x,ly) : NEXT x
  FOR y= INT(y1) TO INT(y2-0.4) : LINE(-lx,y)-(lx,y) : NEXT y
END SUB

SUB gitter (x1,y1,x2,y2)
  FOR x=INT(x1) TO INT(x2)
    FOR y=INT(y1) TO INT(y2)
      PSET (x,y)
    NEXT y
  NEXT x
END SUB
```

12) Teste das Programm. Zeichne im Direktmodus einige Punkte und Strecken ein. Versehe die Achsen noch mit Pfeilspitzen indem du die Prozedur **achsenkreuz** erweiterst. Verwende dazu die Größen lx und ly in dieser Prozedur.

13) Sicher möchtest du auch noch die Achsen beschriften, also ´x´ bzw. ´y´ neben die Pfeilspitzen schreiben. Dies ist allerdings in Q-BASIC nicht ganz einfach, man muss nämlich ausrechnen in welche Zeile und in welche Spalte der Buchstabe geschrieben werden soll. Dann kann man mit LOCATE den Cursor an diese Stelle setzen und mit PRINT den Buchstaben schreiben. Auf der Diskette ist die Lösung dieses Problems zu finden. Eine andere Möglichkeit wäre, durch Ausprobieren die richtige Stelle zu suchen. Also etwa

```
LOCATE 20, 70 : PRINT "x"
```

Punkte können wir mit der Anweisung **PSET** markieren. Allerdings fallen diese Punkte nicht genügend auf, vor allem wenn wir mit Gitterpunkten arbeiten. Deshalb wollen wir ein kleines Kreuzchen an die Stelle des Punktes zeichnen, etwa in Rot (Farbe Nr. 12).

```
SUB kreuz(x,y)
  LINE (x-0.1,y-0.1)-(x+0.1,y+0.1),12
  LINE (x-0.1,y+0.1)-(x+0.1,y-0.1),12
ENDSUB
```

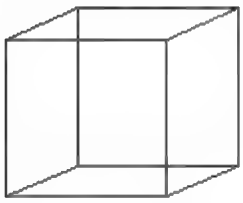
Nimm diese Prozedur noch in das Programm auf. Du kannst sie im Direktmodus erproben. Jetzt hast du alle Möglichkeiten, um auf den Bildschirm so zu zeichnen, wie du es auch sonst in deinem Mathematikheft tust.

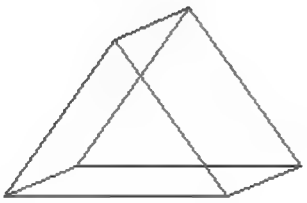
14) Der Benutzer soll geradlinig begrenzte Figuren in das Koordinatensystem zeichnen können. Natürlich kann er das im Direktmodus, wenn er die Namen der entsprechenden Zeichenbefehle weiß. Das Programm soll jetzt aber benutzerfreundlicher werden und zunächst die Anzahl der Ecken, dann die Koordinaten der Eckpunkte abfragen und anschließend die Figur auf den Bildschirm zeichnen. Versuche zunächst die einfachste Möglichkeit: Die Abfragen geschehen noch auf dem normalen Textschirm, dann wird mit SCREEN auf den Grafikschirm umgeschaltet und die Zeichnung erstellt.

15) Benutzerfreundlicher wäre es, wenn der Benutzer schon während der Eingabe von Punkten die Zeichnung verfolgen könnte. Dazu muss also der Text auf einem Teil des Bildschirms Platz finden, die Zeichnung auf einem anderen Teil. Für diesen Zweck gibt es die Anweisung VIEW. Eine andere Möglichkeit wäre, den Text mit LOCATE 1,1 immer wieder am oberen Rand zu platzieren.

16) Die Figur der vorigen Aufgaben soll parallel verschoben werden. Das Programm erfragt die Größe der Verschiebung in x- und y-Richtung und führt dann die Verschiebung aus.

17) Lasse auch noch die Spur der Verschiebung zeichnen. Verzichte auf die Zeichnung des Achsenkreuzes. Auf diese Weise kannst du Schrägbilder von einfachen Körpern zeichnen.





18) Lehrer Matke schreibt für seine Schülerinnen und Schüler ein "Spiel". Auf dem Bildschirm erscheint ein Achsenkreuz und ein zufällig ausgewählter Punkt (Kreuzchen) mit ganzzahligen Koordinaten. Der Schüler wird vom Programm nach diesen Koordinaten gefragt. Anschließend wird die Antwort mit „Richtig“ oder „Falsch“ bewertet.

19) Das obige Programm ist noch zu umständlich; für jeden neuen Rateversuch muss das Programm neu gestartet werden. Wir arbeiten besser mit einer UNTIL-Schleife. Am bequemsten ist es, bei jedem Durchlauf ein neues Achsenkreuz zeichnen zu lassen. Eine andere Möglichkeit wäre, das alte Kreuzchen mit der Farbe Nr. 0 auszuradieren und das alte Achsenkreuz weiter zu verwenden.

20) Bis jetzt konnte das Spiel wohl nur Mathematiklehrer begeistern. Es wird spannender, wenn wir die Spieler unter Zeitdruck setzen. Das Programm zeigt also das Kreuzchen nur noch kurz, radiert es wieder aus und fragt dann nach den Koordinaten. Die Wartezeit können wir durch die SLEEP-Anweisung erzeugen. Wenn der Computer 2 Sekunden warten soll, bevor er weiterarbeitet, geben wir im Programm ein:

```
SLEEP 2
```

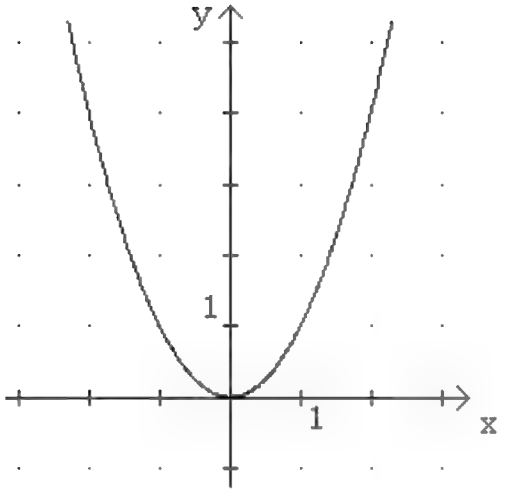
Schön wäre es, wenn der Benutzer die Wartezeit selbst wählen könnte. Vielleicht erwacht dann sein Ehrgeiz.

3.2 Funktionsgraphen

Rechts siehst du den Graph der Quadratfunktion, die Normalparabel. Dazu wurde das Programm zum Zeichnen des Koordinatenkreuzes (Aufgabe 12) erweitert um die Zeilen

```
s=(xmax-xmin)/1280
FOR x= xmin TO xmax STEP s
  y=f(x)
  PSET (x,y)
NEXT x

FUNCTION f(x)
  f = x*x
END FUNCTION
```



Du siehst: Wir zeichnen alle Punkte (x,y), für die y = f(x) gilt. Bei der Schrittweite **s** der **FOR**-Schleife richten wir uns nach der Anzahl der Pixel in x-Richtung.

21) Zeichne ein Schaubild der Funktion f mit
   a) $f(x)= 2x^2+3$
   b) $f(x)= 2(x+5)^2-3$
   Wähle jeweils ein passendes Koordinatensystem.

22) Zeichne Schaubilder der *Standardfunktionen* INT, SGN, ABS, SIN und COS.

23) Zeichne ein Schaubild der Funktion SQR. Vorsicht, SQR(x) ist für negative x nicht erklärt.

24) Durch geeignete Wahl der Grenzen xmin, xmax, ymin, ymax können wir bestimmte Gebiete "mit der Lupe" betrachten. Sieh dir den Verlauf der Quadratwurzelfunktion in der Umgebung des Koordinatenursprungs mit wachsenden Vergrößerungen an.

25) Zeichne das Schaubild der Funktion f mit $f(x) = 1/x$ für $0.01 < x < 20$. Für kleine x-Werte verläuft die Kurve sehr steil, dadurch sind die gezeichneten Punkte weit voneinander entfernt. Das Schaubild wird schöner, wenn wir diese Punkte durch Strecken verbinden:

```
s=(xmax-xmin)/640
PSET (0.01, f(0.01))
FOR x=0.01 TO 20 STEP s
  LINE -(x,f(x))
NEXT x
```

26) Schreibe ein Programm, das nach Eingabe der Steigung m und des y-Achsenabschnitts n die Gerade mit der Gleichung

$$y = mx+n$$

in das Koordinatensystem einzeichnet.

27) In der Aufgabe 87 von Kapitel 2 (Schachtelproblem) suchten wir den Maximalwert einer Funktion. Zeichne das Schaubild dieser Funktion und entscheide die Frage mit Hilfe der Zeichnung.

Nullstellen von Funktionen

Meistens lösen wir Gleichungen rechnerisch. Manchmal fehlen uns aber dazu die nötigen Kenntnisse. Wir suchen z.B. die Lösung der Gleichung

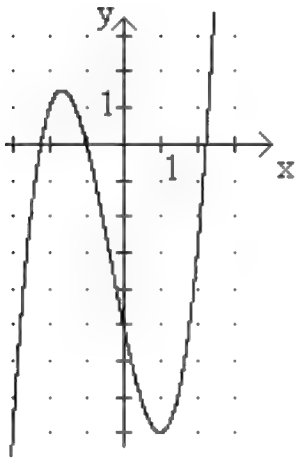
$x^3+x^2-5x-5 = 0.$

Um die Lösungen zumindest näherungsweise zu finden, betrachten wir den Graph der Funktion

$f(x) = x^3+x^2-5x-5$

Dort wo der Graph die x-Achse schneidet, befinden sich die gesuchten x-Werte. Aus der Zeichnung erkennen wir, dass ungefähr gilt

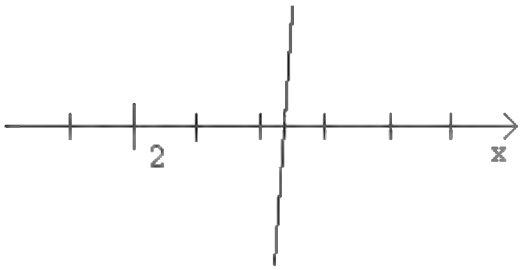
$x_1 = -2$
$x_2 = -1$
$x_3 = 2$



28) Um den Wert x_3 sicher auf eine Stelle nach dem Komma angeben zu können, wählen wir den x-Bereich etwa zwischen 1.9 und 2.5 und ergänzen die Skalenstriche durch eine Zehntel-Teilung. Wir lesen ab

$x_3 = 2.2$

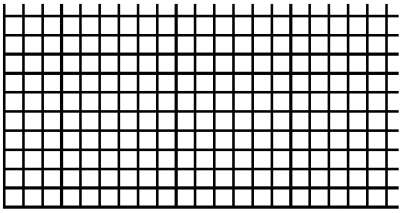
Fahre so fort und bestimme den x-Wert noch genauer.



3.3 Muster der Teilbarkeit

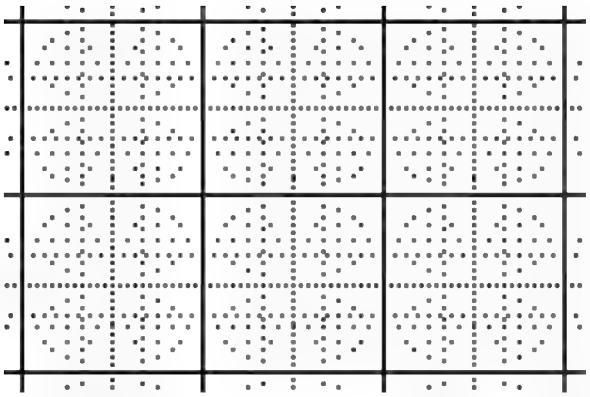
Wir wollen auf dem Bildschirm ein schönes, regelmäßiges Muster erzeugen. Eine allgemeine Methode besteht darin, den Bildschirm Pixel für Pixel durchzugehen und jeweils zu entscheiden, welche Farbe dieser Punkt haben soll. Ein einfaches Beispiel:

```
' Tapetenmuster
SCREEN 12
WINDOW(0,0)-(639,479)
t=7  ' Teiler
FOR x=0 TO 639
 FOR y=0 TO 479
  IF x*y MOD t =0 THEN PSET (x,y)
 NEXT y
NEXT x
```



Das entstehende Muster ist zu regelmäßig, um als besonders schön empfunden zu werden; immerhin ist es eine schöne Veranschaulichung der Tatsache, dass ein Produkt x*y genau dann durch 7 geteilt werden kann, wenn x oder y durch 7 teilbar ist.

29) Durch Verändern von t kannst du beliebig viele verschiedene Muster erzeugen. Rechts siehst du das Muster, das für t=48 entsteht. Kannst du schon voraussagen, was bei t=49 herauskommt?



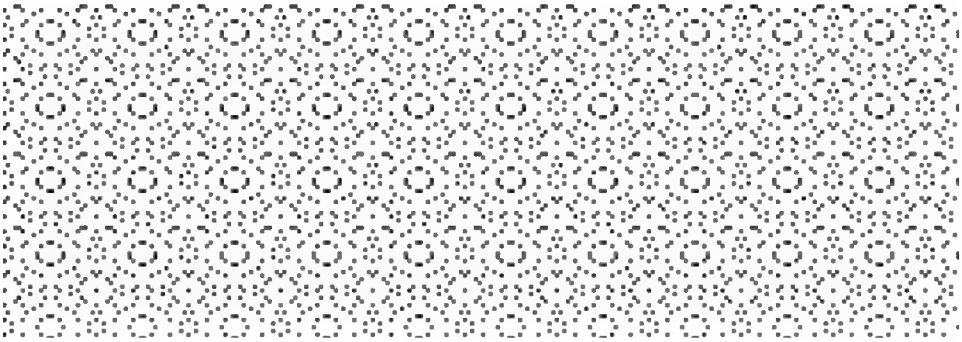
30) Wenn du mit den verschiedenen Teilern genügend experimentiert hast, verändere die entscheidende Zeile, z.B.

```
IF (x+y) MOD t=0 THEN PSET (x,y)
```

Das Ergebnis ist vom ästhetischen Gesichtspunkt enttäuschend. Schönere Bilder ergibt

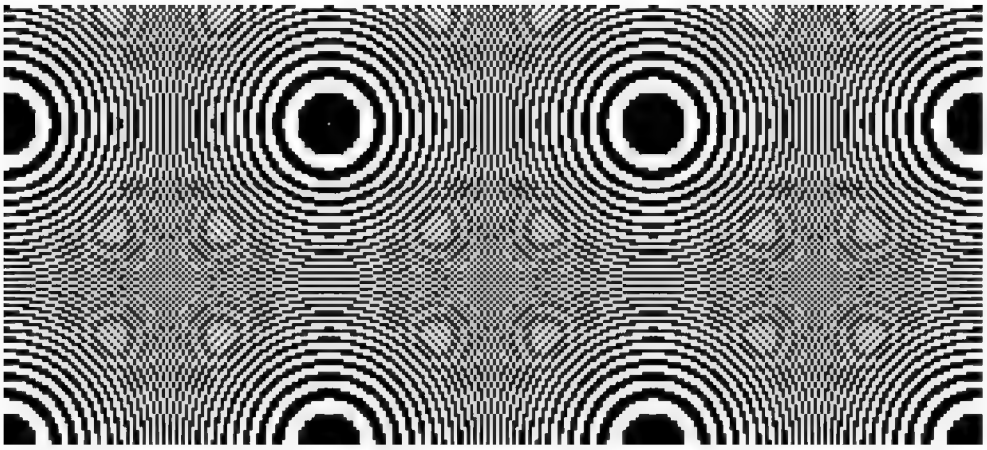
```
IF (x*x+y*y) MOD t=0 THEN PSET (x,y)
```

Das hier gezeigte Muster entsteht für t=25. Andere Teiler (z.B. t=7) ergeben sehr schlichte Muster. Es ist eine reizvolle Aufgabe herauszufinden, welche Teiler einfache Muster liefern.



31) Die bisherigen Muster sind alle recht "dünn". Dies kannst du ändern, indem du beim Teilen durch t nicht nur den Rest 0 zulässt:

```
IF (x*x+y*y) MOD t < t/2 THEN PSET (x,y)
```



32) Das obige Bild entstand mit t=200. Was ergibt sich für größere Werte von t, was für sehr kleine? Falls du einen Farbmonitor hast, werden die Bilder noch schöner. Willst du z.B. in 5 Farben zeichnen, so setze

```
r = (x*x+y*y) MOD t
farbe = INT(5*r/t)
PSET (x,y),farbe
```

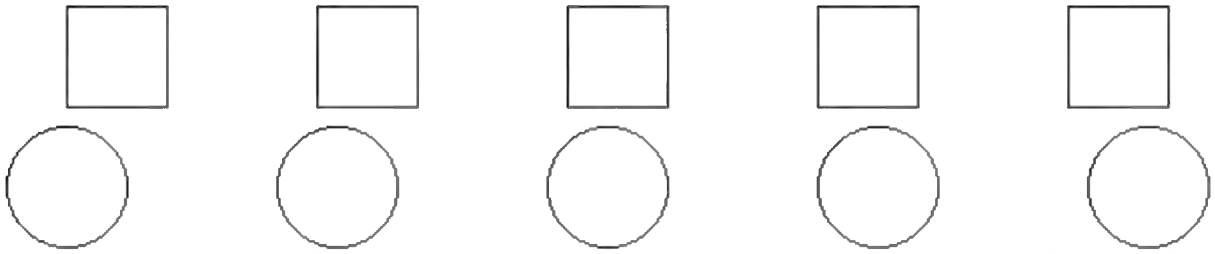
Zunächst wird der Divisionsrest bestimmt, je nach Größe wird diesem eine der 5 Farben zugeordnet.

3.4 Magic-Eye-Bilder

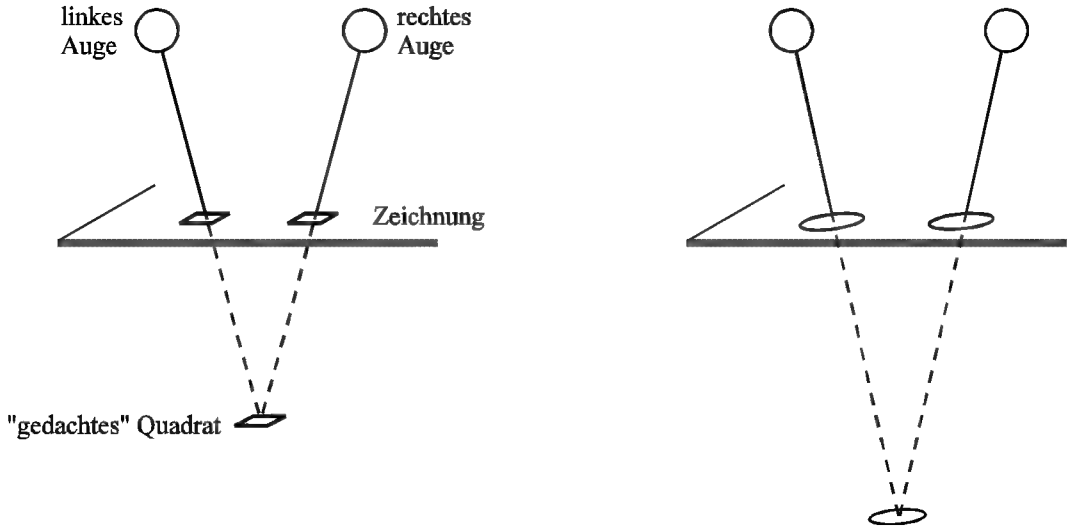
Hast du Freude am Betrachten von so genannten Magic-Eye-Bildern? Dann macht es dir vielleicht auch Spaß, einfache Bilder dieser Art selbst zu erzeugen. Hier ist ein Programm, das ein solches Bild zeichnet:

```
SCREEN 12
WINDOW (0, 0) - (639, 479)
FOR i = 1 TO 5
  CIRCLE (i*110, 190), 30
  LINE (i*100, 100) -(i*100+50, 150),,B
NEXT i
```

Das Programm zeichnet eine Reihe von Kreisen, deren Mittelpunkte einen Abstand von 110 Pixeln haben und eine Reihe von Quadraten, deren Mittelpunkte einen Abstand von 100 Pixeln haben.



Schaust du dieses Bild mit dem Magic-Eye-Blick an, so scheinen die Kreise weiter entfernt zu sein als die Quadrate. Wie ist das zu erklären? Der Trick dabei ist, dass unser linkes Auge ein anderes Quadrat als das rechte Auge anblickt, und dass unser Gehirn dadurch den Eindruck erhält, es handele sich um ein weiter entferntes Quadrat. Aus den Skizzen kannst du auch erkennen, dass die Kreise dann noch weiter entfernt erscheinen.

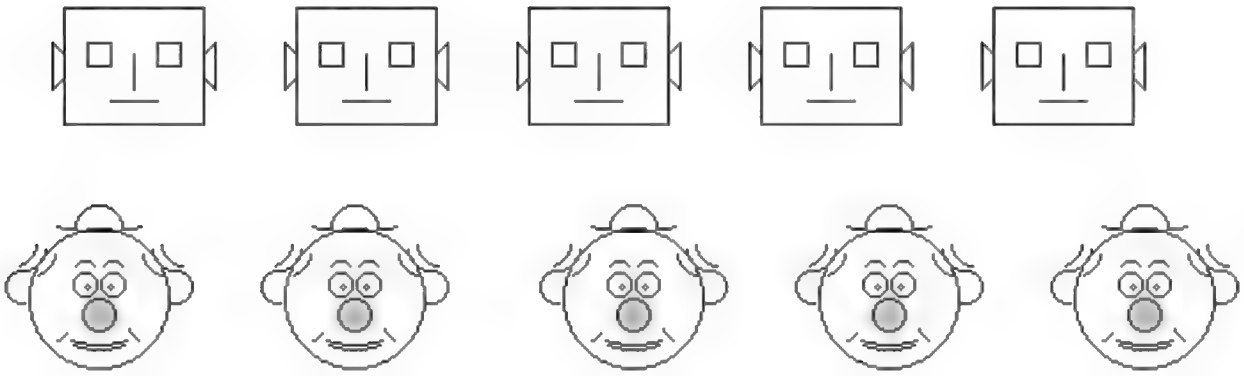


Der Magic-Eye-Blick ist hier ein leichtes „Nach-Außen-Schielen“. Eine andere Möglichkeit, das Bild zu betrachten, ist das „Nach-Innen-Schielen“. Hierbei dreht man die Augen so weit einwärts, dass zwei benachbarte Quadrate zur Deckung kommen. Dann erscheint alles näher zu liegen, die Kreise erscheinen jetzt näher als die Quadrate. Diese Art des Schielens lässt sich unterstützen, indem man das Bild über die Spitze eines Bleistifts anschaut und dabei die Bleistiftspitze fixiert. Die ungefähre Lage der Bleistiftspitze kann man zunächst bestimmen, indem man abwechselnd das linke und das rechte Auge schließt und die Lage des Bleistifts solange verändert, bis er von einem Quadrat zum nächsten zu hüpfen scheint.

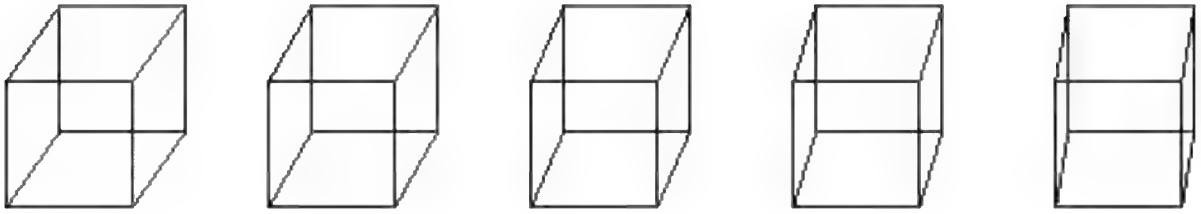
33) Erzeuge mit dem obigen Programm ein Bild, drucke es aus (möglichst Querformat) und betrachte es. Verändere das Bild, indem du mehr Reihen zeichnest und indem du die Abstände veränderst. Wenn sich die Kreise mit den Quadraten überschneiden, ist das Bild besonders gut zu betrachten. Mit Innenschielen kannst du die Bilder auch direkt auf dem Bildschirm anschauen.

34) Die Bilder werden lustiger, wenn du statt Quadraten und Kreisen etwas anderes zeichnest, etwa Roboter- und Clowngesichter. Schreibe dazu eine Prozedur zum Zeichnen eines Robotergesichtes, die von x und y abhängt.

35) Bis jetzt hatten die Objekte einer Reihe immer denselben Abstand. Wenn du diesen Abstand leicht veränderst, gibt es interessante Effekte beim Anschauen.



36) Das folgende Bild ist entstanden, indem zwei Reihen von Quadraten gezeichnet wurden, wobei die Abstände bei den oberen Quadraten etwas kleiner gewählt wurden. Dann wurden entsprechende Ecken miteinander verbunden und so entstand eine Reihe von Schrägbildern eines Würfels. Gleichzeitig ist das ein Magic-Eye-Bild, die Würfel kann man räumlich sehen, besonders gut mit der Methode des Innenschielens. Noch besser wird der Eindruck, wenn man die hinteren Quadrate etwas kleiner zeichnet.

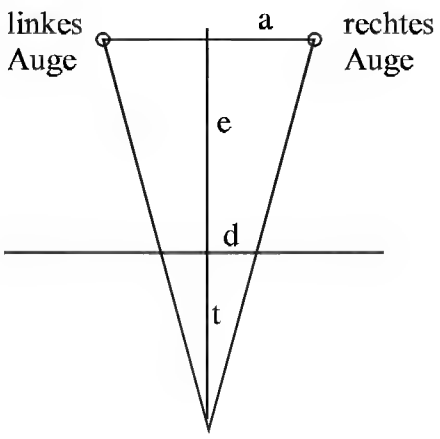


Bei den bisherigen Bildern sah man sofort die dargestellten Objekte, der Magic-Eye-Blick ergab zusätzlich einen räumlichen Eindruck. Du kennst wahrscheinlich noch eine andere Art von Magic-Eye-Bildern, die weitaus überraschender ist: Wenn man das Bild zunächst normal anschaut, kann man gar nichts erkennen; erst das Innen- oder Außenschielen macht ein räumliches Gebilde sichtbar. Zum Beispiel kann man im Bild am Ende dieses Abschnittes eine Halbkugel sehen. Es ist um einiges aufwendiger, solche Bilder zu erzeugen; hier sollen nur die Grundgedanken erläutert werden, sodass es möglich ist, das zugehörige Programm im Anhang zu verstehen.

Wir überlegen uns die Zusammenhänge für den Fall des Auswärtsschielens und beziehen uns dabei auf die Zeichnung zu Beginn dieses Abschnitts. An die Stelle der Quadrate treten jetzt allerdings einzelne Punkte. Um ein „gedachtes“ Bild eines Punktes in einer Tiefe t zu erzeugen, müssen den beiden Augen zwei gleichfarbige Punkte in geeignetem Abstand angeboten werden.

Aus der nebenstehenden Skizze geht der Zusammenhang zwischen der Tiefe t und dem Abstand der Punkte hervor. Wir bezeichnen den halben Abstand der Punkte mit d, den halben Augenabstand mit a, und die Entfernung der Augen vom Papier mit e. Nach dem Strahlensatz gilt dann:

$$d = a \cdot \frac{t}{t+e}$$

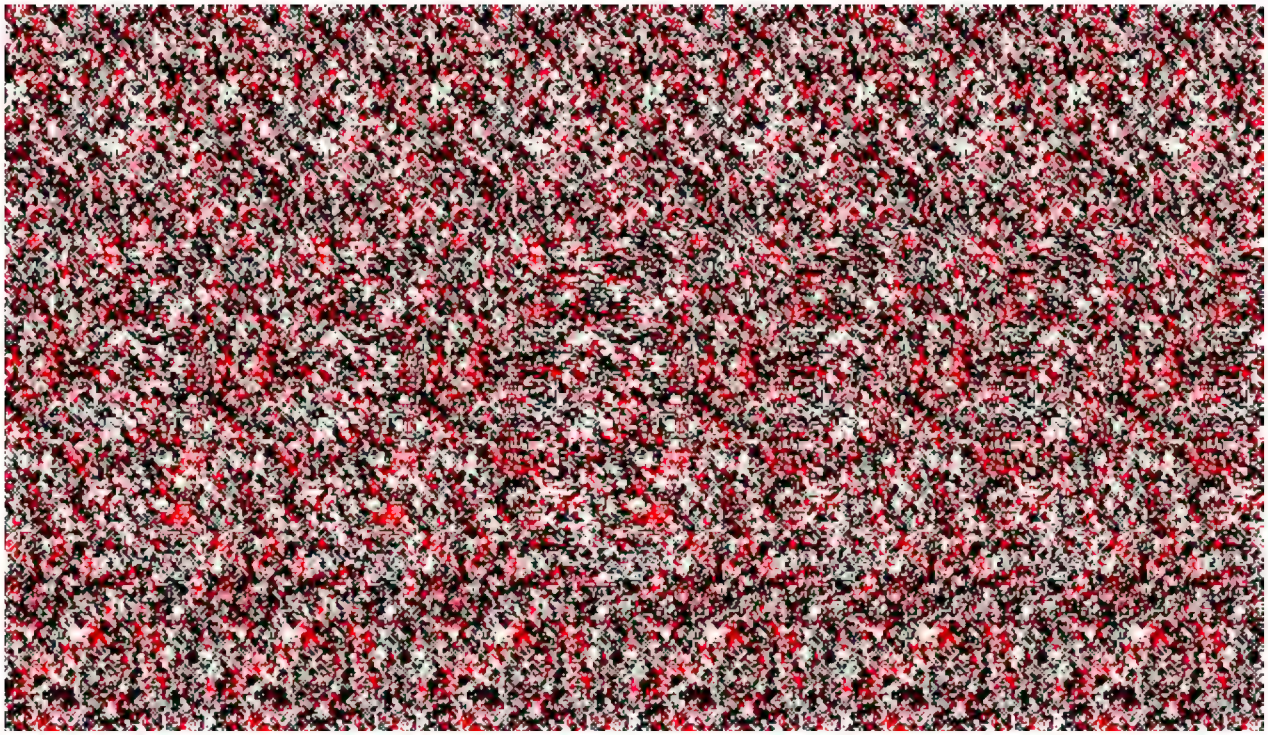


Wenn wir also an der Stelle (x,y) des Bildschirms einen Eindruck der Tiefe t erwecken wollen, müssen die Pixel an den Stellen (x-d, y) und (x+d,y) dieselbe Farbe haben.

Die Einfärbung des Bildschirms geschieht Zeile für Zeile. Zunächst berechnen wir, welche Pixel dieselbe Farbe haben müssen. Dies notieren wir in einem Feld (siehe Kapitel 6). Wir arbeiten dabei von links nach rechts. Ganz links ist die Farbe noch frei wählbar, weil die entsprechenden linken Punkte außerhalb des Bildschirms liegen. Kommt man weiter nach rechts, so muss ein Punkt die gleiche Farbe haben, wie gewisse Punkte, die weiter links liegen.

Die Punkte, deren Farbe frei wählbar ist, können rein zufällig gefärbt werden oder nach einem gewissen Muster; auf der Diskette befinden sich einige Beispiele. Das unten stehende Bild ist mit der folgenden Tiefenfunktion erzeugt.

```
FUNCTION tiefe(x,y)
 r2= (x - 300)^2+(y - 200)^2
 IF r2<10000 THEN
  tiefe = 10 + 0.05*SQR(1000 - r2)
 ELSE
  tiefe=10
 END IF
END FUNCTION
```



4 IGELGRAFIK

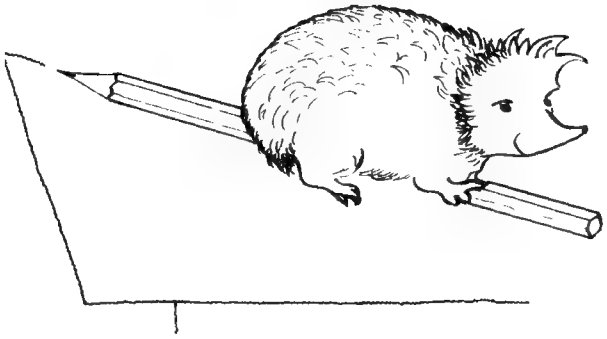
4.1 IGELBEFEHLE IM DIREKTMODUS

Lade das Programm *'igel'* von der Diskette zu diesem Buch. (Voraussetzung ist, dein Computer verfügt über die sog. VGA-Grafik. Falls dein Computer eine EGA-Grafikkarte hat, lade *'igelega'*.) Im Anhang des Buchs findest du ein Listing mit Erläuterung, du kannst das Programm also auch eintippen. Nach dem Laden findest du im Hauptmodul im Wesentlichen nur DIM-Anweisungen, welche die dort genannten Variablen 'global' machen, d.h. jede Prozedur des Programms hat Zugriff zu diesen Variablen und den Prozeduraufruf **igel**. Ansonsten besteht das Programm aus einer Sammlung von Prozeduren. Mit F2 kannst du dir einen Überblick über diese Prozeduren verschaffen: es sind dies **igel**, **re** ('rechts') **li** ('links'), **vw** ('vorwärts'), **rw** ('rückwärts'), **sa** ('stiftab') und **sh** ('stifthoch').

Gib im Direktmodus ein:

```
igel
```

Der Befehl zaubert auf den Grafikschirm ein kleines Dreieck mit einer 'Nase', wir sprechen dieses Dreieck liebevoll als 'Igel' an. Sein Erfinder in den USA hat es 'turtle' ('Schildkröte') genannt - in den USA gibt es keine Igel. Du bist jetzt in der sog.
Igelgrafik.



1) Gib nacheinander folgende Befehle im Direktmodus ein:

```
vw 40
re 90
vw 200
rw 300
li 110
vw 150
re 120
vw 100
```

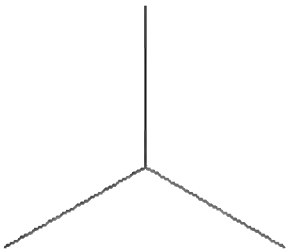
Du siehst: Der Igel ist dein Zeichenstift, den du mit den Befehlen **vw**, **rw**, **li** und **re** über den Bildschirm dirigieren kannst. Etwas lästig ist das ständige Umschalten zwischen Textschirm zur Eingabe der Befehle und Grafikschirm, zur Ausführung jeden Befehls.

2) Der Bildschirm ist jetzt ganz schön vollgemalt. Gib diesen Befehl:

```
igel
```

3) Die folgende Befehlsfolge zeichnet einen dreistrahligen Stern:

```
vw 100 : rw 100 : re 120
vw 100 : rw 100 : re 120
vw 100 : rw 100 : re 120
```



Du kannst mehrere Igelbefehle, getrennt durch einen Doppelpunkt, in eine Zeile schreiben. Du brauchst nur eine der drei Zeilen wirklich tippen, gehe anschließend mit CURSORHOCH in die Zeile zurück und drücke die EINGABETASTE.)

4) Rufe wieder den Igel auf und zeichne einen sechsstrahligen Stern.

5) Probiere aus, wie weit der Igel gehen kann, ohne den Bildschirm zu verlassen. Gib ein:

```
igel
vw 100
vw 100
```

Taste dich so an den Rand vor. Beachte: von der Bildschirmmitte sind es etwa 200 "Igelschritte" nach oben und unten und 300 "Igelschritte" nach links und rechts.

6) Probiere nun folgende Befehlsfolge aus:

```
igel
sh
rw 50 : li 90 :vw 50 :re 90 :sa
vw 100 : re 90   (4-mal wiederholen)
```

Was bewirken die Befehle **sh** und **sa**?

Anmerkung: Wir haben den Befehl **sh** nicht mit Doppelpunkt getrennt an den Anfang der (jetzigen) dritten Zeile geschrieben, denn Q-BASIC fasst eine Buchstabenfolge gefolgt von einem Doppelpunkt als 'Marke' (eine Sprungadresse) auf und nicht als Prozeduraufruf!

In der nachstehenden Tabelle sind unsere Igelbefehle zusammengefasst:

`igel`	ruft die 'Igelgrafik' auf, zeichnet den Igel in die Bildschirmmitte. Erster Befehl in jedem Igelprogramm.
`vw 100`	vorwärts 100, Igel geht um 100 'Schritte' (pixels) vorwärts (in Richtung der 'Nase').
`rw 50`	rückwärts 50, Igel geht 50 pixels zurück.
`re 90`	rechts 90, Igel dreht sich auf der Stelle um 90° nach rechts.
`li 30`	links 30, Igel dreht sich auf der Stelle um 30° nach links.
`sh`	stifthoch, die nachfolgenden Igelbewegungen werden nicht aufgezeichnet.
`sa`	stiftab, die nachfolgenden Igelbewegungen werden aufgezeichnet.

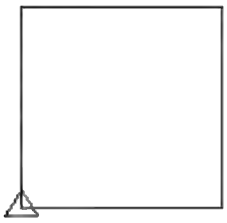
4.2 Vielecke

Nachdem du die Igelbefehle kennen gelernt hast, wollen wir einige Programme zur Steuerung des Igels schreiben. Das 'Igelprogramm' (d.h. die Unterprogramme, welche die Igelbefehle erklären) muss stets geladen sein. Du schreibst deine Programme ins 'Hauptmodul' am Ende des von der Diskette geladenen 'Rumpfprogramms' oder in entsprechende Unterprogrammfenster. Im Hauptmodul müssen mindestens die DIM-Anweisungen stehen und die Igelgrafik aktiviert werden. Jedes Programm beginnt also mit

```
DIM SHARED ..
...
DIM SHARED ..
igel
```

Diese Zeilen lassen wir in Zukunft weg.

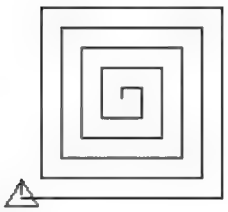
7) Wir wollen, dass der Igel ein Quadrat zeichnet. Was müssen wir ihm dazu befehlen? Versetz dich in die Rolle des Igels. Offenbar musst du zuerst eine Strecke, etwa 100 Schritte, vorwärts gehen, dich auf der Stelle um 90° nach rechts (oder links) drehen, wieder dieselbe Strecke vorwärts gehen, dich um 90° drehen usw. Gib also ein und führe aus:



```
' Quadrat
FOR i=1 TO 4
 vw 100: re 90
NEXT
```

Der Igel hat seine Arbeit so schnell erledigt, dass du nur noch das fertige Quadrat bewundern kannst. Bei der nächsten Aufgabe wirst du ihm aber zuschauen können. Er soll nämlich eine (quadratische) Spirale zeichnen. Was müssen wir ihm dazu sagen? Doch etwa dieses:

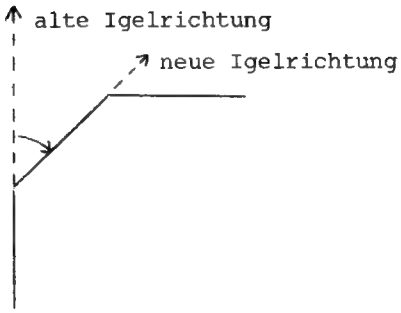
Gehe zuerst eine (kleine) Strecke s voran, dann drehe dich um 90° (nach rechts), gehe nun eine etwas längere Strecke voran, drehe dich um 90°, gehe eine gegenüber vorher wieder etwas längere Strecke voran, drehe dich um 90° usw.



8) Gib nachstehendes Programm ein und überzeuge dich, dass es die Aufgabe löst (markiere zuerst die Zeilen des vorigen Programms und entferne sie mit ENTF oder lade das Programm 'igel' neu).

```
' Spirale
FOR s= 5 TO 200 STEP 2
 vw s: re 90
NEXT
```

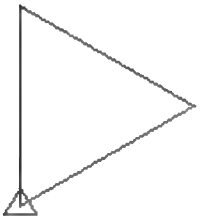
Jetzt siehst du den Igel über den Bildschirm flitzen und die gewünschte Spirale zeichnen.



9) Du möchtest ein gleichseitiges Achteck zeichnen. Im nachstehenden Programm fehlt nur der Winkel, um den du den Igel in den Ecken drehen musst. Ergänze diesen Winkel, die Zeichnung hilft dir dabei.

```
' Achteck
FOR i=1 TO 8
 vw 100: re
NEXT
```

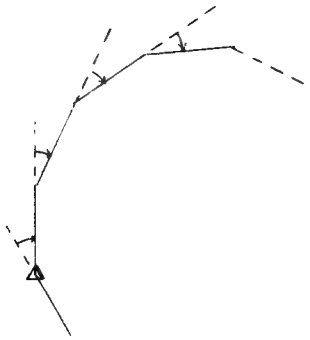
10) Nach den bisherigen Aufgaben wird dir die nächste lachhaft leicht vorkommen. Du sollst den Igel nämlich veranlassen ein gleichseitiges Dreieck zu zeichnen. (Aufgepasst! Versetz dich in den Igel: um wie viel Grad muss er sich in jeder Ecke drehen?)



Wir wollen den Igel ein regelmäßiges n-Eck zeichnen lassen. Die Eckenzahl und die Seitenlänge sollen vom Benutzer erfragt werden. An jeder Ecke dreht sich der Igel um denselben Winkel. Wie groß ist dieser Drehwinkel?

Kehrt der Igel nach Durchlaufen des ganzen Vielecks zu seiner Ausgangsstelle zurück, hat er sich einmal um sich selbst gedreht, d.h. um 360°. Diese 360°-Drehung verteilt sich nun auf die Drehungen in den einzelnen Ecken.

Zeichnet der Igel ein 5-Eck, so dreht er sich in jeder Ecke um 360°/5, zeichnet er ein 6-Eck, dreht er sich in jeder Ecke um 360°/6, usw.



```
' Vieleck
PRINT "Regelmäßiges Vieleck"
INPUT "Wie viele Ecken? ", anzahl
INPUT "Seitenlänge? ", seite
winkel=360/anzahl
FOR i=1 to anzahl
 vw seite: re winkel
NEXT
```

11) Gib das oben stehende Programm ein und teste es für verschiedene Eckenzahlen und Seitenlängen.

Du stellst fest: wächst die Eckenzahl, so nähert sich das gezeichnete n-Eck der Kreisform. Infolge der begrenzten Auflösung des Bildschirms ist z.B. ein 60-Eck von einem Kreis nicht zu unterscheiden.

12) Wir kehren noch einmal zu unserer Spirale zurück.
a) Ändere die Drehwinkel im Programm 'Spirale' in 72° (60°, 120°). Was für Spiralen entstehen jeweils?
b) Interessante Figuren entstehen , wenn der Drehwinkel nahe bei einem "Vieleckswinkel" liegt. Wähle z.B. 89°, 91°, 119°, 73° usw.
c) Experimentiere mit weiteren Drehwinkeln. Probiere Winkel aus, die kein Teiler von 360 sind aber mit 360 einen größeren gemeinsamen Teiler haben, z.B. 144°, 135°, ..

4.3 Prozeduren

Die Aufgabe, ein Quadrat zu zeichnen, kommt in der Geometrie häufig vor. Anstatt jedes Mal das Programm von Aufgabe 7 eintippen zu müssen, schreiben wir ein Unterprogramm (eine Prozedur) **quadrat**, die bei Aufruf durch ihren Namen ausgeführt wird.

13) Gib dieses Unterprogramm ein:

```
SUB quadrat
  FOR i=1 to 4
   vw 100: re 90
  NEXT
END SUB
```

a) Teste die Prozedur im Direktmodus:

```
igel
quadrat
```

b) Gib dieses kleine Programm ein:

```
FOR k=1 TO 8
  quadrat
  re 45
NEXT
```

Unsere Prozedur **quadrat** hat einen entscheidenden Mangel: Wir können mit ihr nur Quadrate mit einer festen Seitenlänge (hier 100 Schritte) zeichnen. Wirklich nützlich wird der Befehl erst sein, wenn wir mit ihm Quadrate unterschiedlicher Seitenlänge zeichnen können. Der Befehl **quadrat** soll also noch,

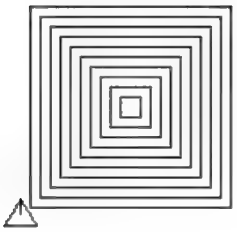
wie unser Igelbefehl **vw**, von einem 'Beiwert', man sagt '*Parameter*' abhängen: **quadrat 120** soll ein Quadrat der Seitenlänge 120 Schritte, **quadrat 70** ein Quadrat der Seitenlänge 70 Schritte zeichnen. Um dies zu erreichen, müssen wir das Unterprogramm **quadrat** mit einer 'variablen' Seitenlänge definieren, erst beim Aufruf bekommt diese Variable dann den im Aufruf genannten Wert zugewiesen.

14) Bearbeite das Unterprogramm **quadrat** (F2 und doppelt anklicken oder 'im aktiven Fenster bearbeiten' anklicken) und ändere es so ab:

```
SUB quadrat(seite)
  FOR i=1 TO 4
   vw seite: re 90
  ENDFOR
END SUB
```

Rufe die Prozedur dann in einem Programm auf:

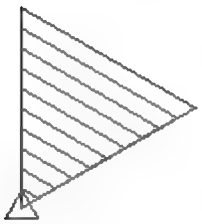
```
FOR k = 1 to 8
  quadrat 20*k
NEXT
```

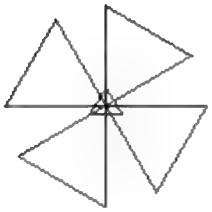


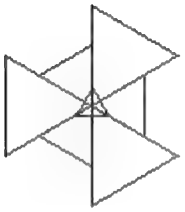
15) Obiges Programm zeichnet die Quadrate aneinander angrenzend. Schreibe ein Programm, das konzentrisch ineinander liegende Quadrate zeichnet (Hinweis: Benutze **sh** und **sa**).

16) Schreibe eine Prozedur **dreieck**, die bei Aufruf gleichseitige Dreiecke mit der im Aufruf genannten Seitenlänge zeichnet. Verwende die Prozedur, um folgende Bilder zu malen:

a) b) c)







4.4 Kreisprozeduren

Wir haben oben gesehen, dass sich ein regelmäßiges 60-Eck auf dem Bildschirm von einem Kreis nicht unterscheidet.

17) Gib das nachstehende Programm ein:

```
' kreis
FOR i=1 TO 60
  vw 5: re 6
NEXT
```

Ersetze `vw 5` durch `vw 3` (`vw 8, vw 10`). Wie ändert sich der gezeichnete Kreis? Welchen Umfang und welchen Durchmesser hat jeder der vier Kreise?

Für den Zusammenhang zwischen Umfang und Radius eines Kreises gilt die berühmte Beziehung:

Kreisumfang = 2∗π ∗ Kreisradius

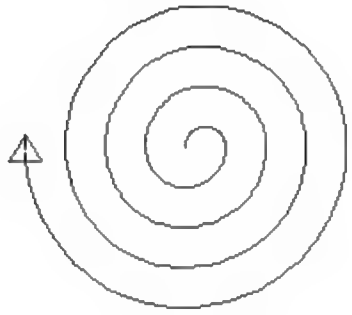
dabei ist die Kreiszahl π rund 3,14.

Unser Ziel ist eine Prozedur **kreis(r)**, welche einen Kreis mit dem Radius r zeichnet. Diesen Kreis realisieren wir als 60-Eck. Um dieses zu zeichnen, müssen wir seine Seitenlänge s kennen. Wie groß ist s? Nun, der Umfang des 60-Ecks ist natürlich 60∗s, als Kreis betrachtet ist der Umfang aber $2\pi r$, also $2\pi r = 60s$ oder $s = 3.14*r/30$. Unsere gewünschte Prozedur kann somit geschrieben werden:

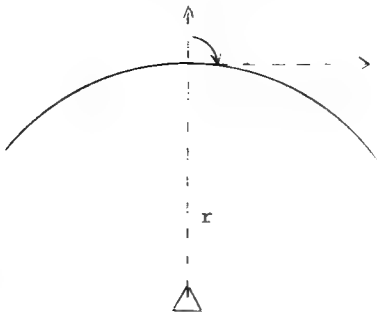
```
SUB kreis(r)
  s=3.14*r/30
  FOR i=1 TO 60
   vw s: re 6
  NEXT
END SUB
```

18) Gib die Prozedur ein und rufe sie im Direktmodus für verschiedene Werte von r auf.

19) Schreibe eine analoge Prozedur **halbkreis(r)** und zeichne mit ihrer Hilfe eine Kreisspirale.



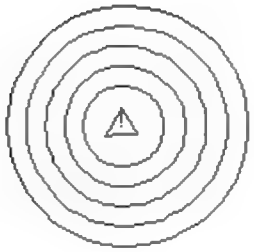
Die Prozedur **kreis(r)** zeichnet einen 'Rechtskreis' vom Radius r. Die Zeichnung beginnt an der Position, wo der Igel gerade steht und in der Richtung, in welche die Nase des Igels weist. Wir wollen unsere Prozedur so abändern, dass die Igelposition beim Prozeduraufruf gerade der Kreismittelpunkt ist. Dazu müssen wir den Igel zuerst ohne zu zeichnen auf den Kreisrand setzen und ihn dann noch in die Kreisrichtung drehen:




```
sh
vw r: re 90: sa
```

Anmerkung: Wir schreiben die vier Befehle nicht durch : getrennt in eine Zeile, da Q-BASIC sonst die Zeichenfolge sh: als Marke ansieht und nicht als Prozeduraufruf.

20) Zeichne mit der Prozedur **kreis(r)**, wie sie jetzt vorliegt, eine Zielscheibe.



Du wirst wahrscheinlich ein Programm geschrieben haben, wie das folgende:

```
' Zielscheibe
FOR r=10 TO 100 STEP 20
  kreis(r)
NEXT
```

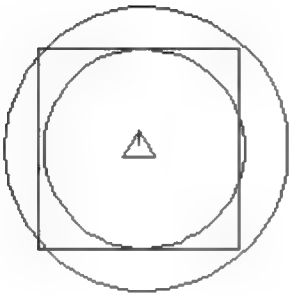
Zu deiner Überraschung entsteht aber keineswegs eine Zielscheibe, die Kreise liegen vielmehr recht wirr auf dem Bildschirm. Du wolltest Kreise mit wachsendem Radius zeichnen, die alle die Bildschirmmitte als Zentrum haben. Offenbar wird die letzte Bedingung nicht erfüllt. Woran liegt das? Die Position des Igels beim Prozeduraufruf ist der Kreismittelpunkt. Nur beim ersten Aufruf kreis(10) ist dies die Bildschirmmitte, danach immer ein Punkt auf dem Umfang des zuvor gezeichneten Kreises. Wir müssen also unsere Prozedur **kreis(r)** noch insofern verbessern, dass sie den Igel in der Position zurücklässt, in welcher sie ihn übernommen hat.

```
SUB kreis(r)
 'zeichnet Kreis mit Radius r. Die Position des Igels
 'beim Prozeduraufruf ist der Kreismittelpunkt.
 'Der Igel wird so hinterlassen, wie er übernommen wurde.
 s=3.14*r/30
 sh
 vw r: re 90: sa
 FOR i=1 TO 60
   vw r: re 6
 NEXT
 sh
 li 90: rw r: sa
END SUB
```

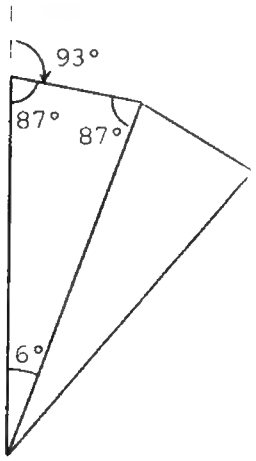
In den Kommentarzeilen ist die *Spezifikation* der Prozedur angegeben, d.h. die genaue Beschreibung der Wirkung der Prozedur.

21) Zeichne mit der verbesserten Kreisprozedur die Zielscheibe.

22) Zeichne ein Quadrat, seinen Inkreis und seinen Umkreis. Anmerkung: Hat das Quadrat die Seitenlänge a, so hat der Umkreis den Durchmesser $a\sqrt{2}$. Für $\sqrt{2}$ kannst du den Wert 1.41 verwenden.



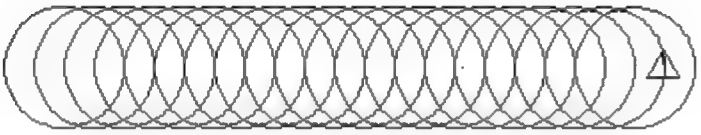
Das Bild, welches bei Lösung von Aufgabe 22 entsteht, befriedigt keineswegs. Irgendwie paßt der Umkreis nicht richtig um das Quadrat herum, der Inkreis nicht richtig in das Quadrat hinein. Welchen Fehler haben wir gemacht?
Wir müssen uns erinnern, dass unser Kreis ja in Wirklichkeit ein 60-Eck ist. Damit das gezeichnete 60-Eck aber den Ausgangspunkt des Igels als Mittelpunkt hat, müssen wir den Igel, wie der Zeichnung zu entnehmen ist, nicht um 90°, sondern um 93° in den Umfang des 60-Ecks hineindrehen. Die entsprechenden Befehle `re 90` und `li 90` sind also zu ersetzen durch `re 93` bzw. `li 93`.



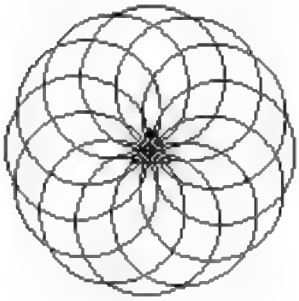
23) Löse Aufgabe 22 mit der verbesserten Kreisprozedur.

24) Schreibe Programme, welche die abgebildeten Kreismuster erzeugen.

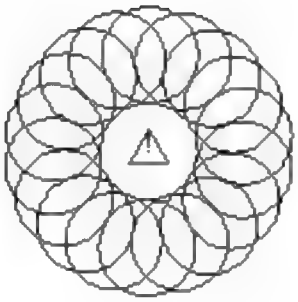
a)



b)

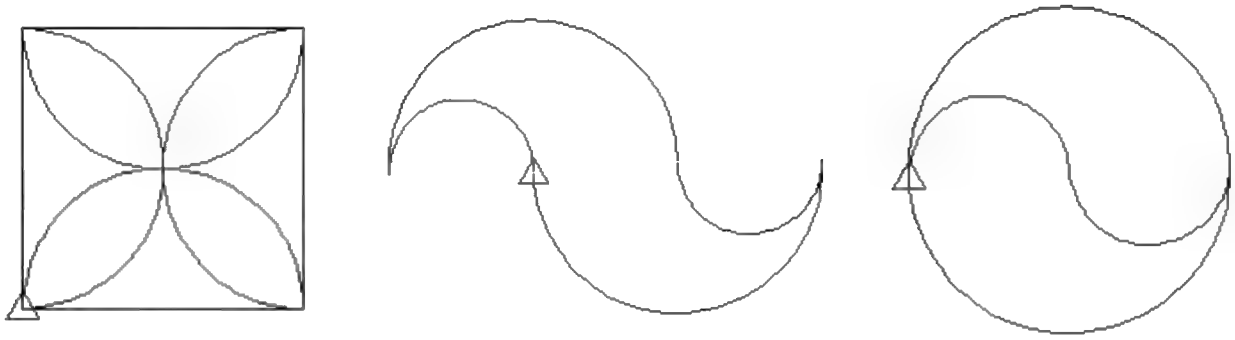


c)



25) Zeichne die nachstehenden Bilder. Verwende dazu eine Prozedur **halbkreis(r)**.

a) b) c)



d) Formuliere die Spezifikation der Prozedur **halbkreis(r)**.

4.5 Projekt Buchstaben

Zum Abschluss wird ein größeres Projekte dargestellt, bei dessen Bearbeitung du Einsicht in die Denkweisen des "strukturierten Programmierens" gewinnst, das aber auch Spaß machen soll.

Wir wollen unsere Namen in ganz großen Buchstaben auf den Bildschirm schreiben.

Die Aufgabe legt eine Arbeitsteilung nahe: Jeder schreibt nur eine Prozedur, die gerade einen Buchstaben erzeugt. Am Ende tragen wir unsere Ergebnisse zusammen. Damit sich die Buchstaben dann auch gut zusammenfügen, sind einige Verabredungen notwendig: Zumindest über Höhe und Breite der Buchstaben müssen wir uns zuerst verständigen.
Es folgt ein Vorschlag für die Buchstaben A und B:

```
SUB ba
  vw 100: re 90: vw 50: re 90: vw 100
  rw 50: re 90: vw 50
END SUB

SUB bb
  vw 100: re 90: vw 25
  halbkreis 25
  vw 25: rw 25: re 180
  halbkreis 25: vw 25
END SUB
```

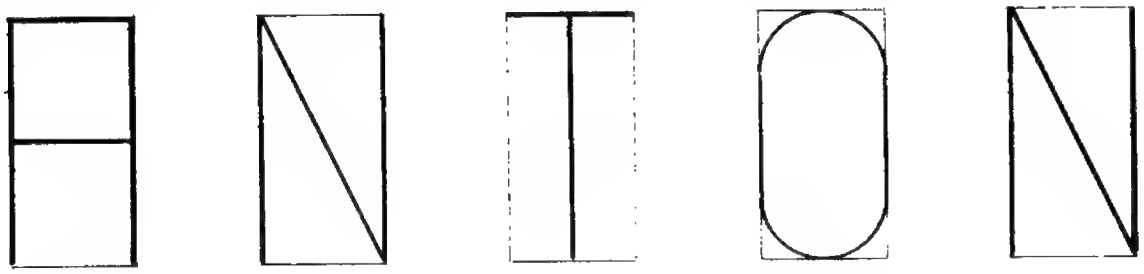
Anmerkung: Wir nennen unsere Prozeduren ba ('Buchstabe a'), bb ('Buchstabe b') usw. weil Q-BASIC gleiche Namen für Variable und Prozeduren nicht erlaubt. Im 'Igelprogramm' kommen aber viele Variable vor, die nur mit einem Buchstaben bezeichnet wurden.
Gib dieses Prozeduren ein. Mit dem Aufruf **ba** bzw. **bb** kannst du nun diese Buchstaben zeichnen. Du stellst schnell fest: Unsere Prozeduren sind noch mangelhaft. Zwar wird jeder Buchstabe für sich allein richtig gezeichnet, das "Zusammenspiel" der Prozeduren klappt aber nicht. Es gelingt uns nicht, ein Wort aus unseren Buchstaben zusammenzusetzen. Der Grund ist auch klar: Jeder Buchstabe hinterlässt den Igel in ganz planloser Weise, sodass der Beginn des nächsten Buchstabens nicht zu überschauen ist. Wir müssen in unsere Verabredung auch mit aufnehmen, wie jede Buchstabenprozedur den Igel zurücklässt. Eine Möglichkeit ist, dass jede Buchstabenprozedur den Igel so hinterlässt, wie sie ihn übernommen hat.

Die Prozedur **ba** muss bei dieser Verabredung dann so ergänzt werden :

```
re 90: rw 50
```

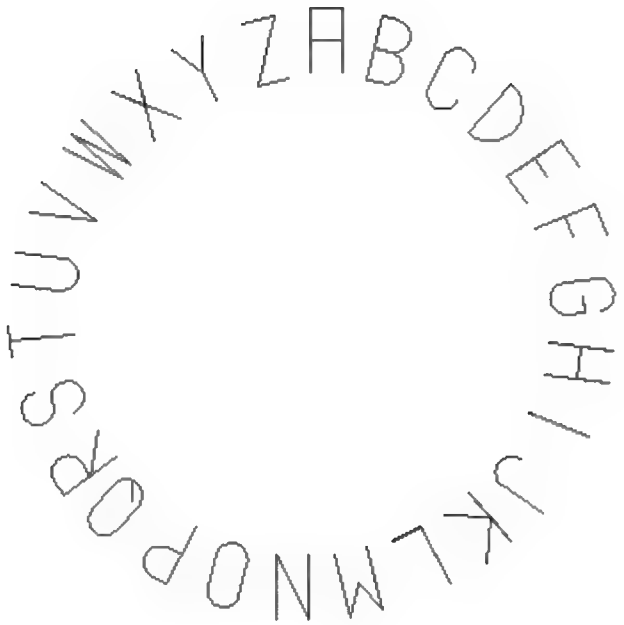
Ergänze entsprechend die Prozedur **bb**.
Schreibe nun noch Prozeduren für die Buchstaben N, T und O und schreibe mit ihnen den Namen Anton in Großbuchstaben.

Wir sind noch keineswegs zufrieden. Wir müssen den Igel nach jedem Prozeduraufruf an den Anfang des nächsten Buchstabens setzen (und der Abstand von A zu N ist auch noch ein anderer wie von N zu T!) Wie können wir uns die Arbeit vereinfachen?
Wir denken uns die Buchstaben immer in ein Rechteck eingeschlossen, etwa so:



Wenn nach Aufruf einer Buchstabenprozedur der Igel immer gleich in der unteren linken Ecke des nächsten Buchstabenrechtecks hinterlassen wird, entfällt das mühsame Umsetzen des Igels und um z.B. das Wort "ANTON" zu schreiben, besteht unser Hauptmodul nur aus den Aufrufen dieser Buchstabenprozeduren. Natürlich muss beim Schreiben der einzelnen Buchstabenprozeduren die Ausgangsstellung des Igels beachtet werden, beim Buchstaben T werden wir also z.B. zuerst den Igel ohne zu zeichnen in die Mitte des unteren Rechtecks bringen.

Versuche mit deinen Buchstabenprozeduren auch den abgebildeten Buchstabenkreis zu erzeugen. (Wie sollte jetzt der Igel nach dem Zeichnen eines Buchstabens hinterlassen werden?) Auf der Begleitdiskette zu diesem Buch findest du die Programme 'alphabet.bas' und 'bkreis.bas'.



5 Textbearbeitung

5.1 Textvariable und Operationen mit Textvariablen

Das folgende Programm schreibt eine Einladung:

```
' Einladung
CLS
PRINT "      Einladung"
PRINT "      =========="
PRINT
PRINT "Lieber Mark,
PRINT
PRINT "Zu meiner Party am Dienstag, den 5. November 1996,"
PRINT "um 18 Uhr"
PRINT "lade ich dich herzlich ein"
PRINT
PRINT "Ulrike"
```

Es wäre schön, wenn Ulrike nicht für jeden Gast, den sie einladen will, das Programm ändern muss. Was Ulrike braucht, ist eine *Variable für den Namen des Eingeladenen*, welcher sie mit einer **INPUT**-Anweisung den jeweiligen Namen zuweisen kann.

1) Ergänze obiges Programm um die folgenden Zeilen und ändere die Anrede wie angegeben. Das Programm schreibt jetzt Einladungen für beliebige Gäste.

```
' Einladung
INPUT "Eingeladen wird: ",gast$
PRINT "Liebe(r)";gast$
```

In Zeile 2 wird der Computer angewiesen, eine "Schublade" mit dem Namen **gast$** bereitzuhalten, in welcher Text gespeichert werden kann. Durch das angehängte Dollarzeichen "$" werden Textvariable von numerischen Variablen unterschieden.

2) Versuche, folgendes Programm auszuführen. Wie reagiert Q-BASIC?

```
a$=234
b="hallo"
PRINT a$;b
```

3) Der Computer soll dich nach deinem Namen fragen und dann den Namen in einer freundlichen Begrüßung verwenden.

4) Der Computer fragt "Programmierst du gerne? (j/n)". Auf die Antwort "j" antwortet er mit "Faszinierend, nicht?", auf die Antwort "n" antwortet er mit "Das ist aber schade."

5) Wie reagiert das nachstehende Programm auf die Antworten "jawohl", "njet", "keinesfalls", "jein"?

```
' antworten
CLS
antw$ = ""
WHILE antw$<>"j" AND antw$<>"n"
 INPUT "Programmierst du gerne? (j/n) ", antw$
 IF antw$ = "j" THEN
  PRINT "Faszinierend, nicht?"
 ELSEIF antw$ = "n" THEN
  PRINT "Schade"
 ELSE
  PRINT "Bitte nur 'j' oder 'n' antworten"
 END IF
WEND
```

Ulrike feiert öfters eine Party und schreibt jedes Mal viele Einladungen. Mit nachstehendem Programm hat sie sich ein für alle Mal die Arbeit des Einladens erleichtert.

```
' Einladungen
CLS
PRINT "Einladungen zur Party"
INPUT "Datum der Party: ", datum$
INPUT "Uhrzeit (hh.mm): ", zeit$
antw$ = "j"
WHILE antw$ = "j"
  PRINT
  INPUT "Eingeladen wird: ", gast$
  druck
  INPUT "Noch eine Einladung?(j/n) ", antw$
WEND
```

Zuerst wird nach Datum und Zeit der Party gefragt, in der WHILE-Schleife (zwischen den Schlüsselwörtern WHILE und WEND) nach dem Namen des Gasts und mit dem Prozeduraufruf "**druck**" die entsprechende Einladung gedruckt. Die Schleife wird verlassen, sobald auf die Frage "Noch eine Einladung?" nicht mit "j" geantwortet wird. Das Drucken der Einladung soll das Unterprogramm (die *Prozedur*) **druck** besorgen.

```
SUB druck
  SHARED gast$, datum$, zeit$
  LPRINT "                Einladung"
  LPRINT "                =========="
  LPRINT
  LPRINT "Lieber  "; gast$
  LPRINT
  LPRINT "Zu meiner Party am "; datum$; " um "; zeit$
  LPRINT "lade ich dich herzlich ein."
  LPRINT
  LPRINT "Ulrike"
  LPRINT
END SUB
```

Nach Eingabe der Schlüsselworte SUB druck (EINGABE-Taste drücken) wechselt der Bildschirm zum "Unterprogrammfenster" und du gibst die Programmzeilen wie gewohnt ein. Mit F2 erscheint ein Dialogbildschirm zum Wechseln der Fenster.

Schauen wir uns die Prozedur **druck** genauer an. Um die erste Zeile zu verstehen muss du wissen: Variable in Unterprogrammen sind in Q-BASIC grundsätzlich "lokal", d.h. nur das betreffende Unterprogramm weiß um sie, andere Unterprogramme und das Hauptprogramm (Q-BASIC spricht vom "Hauptmodul") kennen diese Variable nicht, auch wenn sie gleiche Namen haben sollten. Umgekehrt kann ein Unterprogramm nicht auf Variable im Hauptmodul oder in anderen Unterprogrammen zugreifen. Da die Prozedur **druck** aber vom Hauptprogramm die jeweilige Belegung der Variablen *gast$*, *datum$* und *zeit$* wissen muss, wird der Prozedur mit dem Befehl **SHARED** Zugang zu diesen Variablen geschaffen. Die nächsten Zeilen sind leicht verständlich. Du musst nur wissen, dass LPRINT wie PRINT wirkt, allerdings findet die Ausgabe nicht auf dem Bildschirm sondern auf dem angeschlossenen Drucker statt (vorausgesetzt, er ist an der üblichen Druckerausgabe, der sog. Schnittstelle LPT1 angeschlossen).

6) a) Gib Ulrikes Programm ein und probiere es aus. Falls du keinen Drucker angeschlossen hast, musst du LPRINT durch PRINT ersetzen. (Q-BASIC hilft hierbei: markiere das Wort LPRINT und wähle im Menü SUCHEN den Punkt ÄNDERN...)
   b) Warum ist die Datums- und Zeitabfrage nicht mit in die **WHILE**-Schleife aufgenommen?
   c) Lösche die Zeile mit dem Befehl SHARED im Unterprogramm. Was wird nun gedruckt?

7) Schreibe nach obigem Muster ein Programm, das für dich die Bedanke-mich-Briefe nach Weihnachten schreibt.

Eine wichtige Operation, die man mit Texten ausführt, ist das einfache Aneinanderhängen. (Anstelle des Wortes "Text" verwenden wir auch den Begriff "Zeichenkette" oder das englische Wort "string".)

8) Gib das nachstehende Programm ein und beobachte den Ausdruck.

```
' Verketten1
a$ = "tür": b$ = "flügel"
c$ = a$ + b$: d$ = b$ + a$
PRINT c$
PRINT d$
```

9) a) Welchen Ausdruck erzeugt dieses Programm? Überprüfe deine Vermutung.

```
' Verketten2
a$ = "17": b$ = "25"
x = 17: y = 25
PRINT a$+b$
PRINT x + y
```

   b) Ersetze in den Zeilen 3 und 5 die Variablennamen **x** und **y** durch **a** und **b**. Wie reagiert Q-BASIC?

10) a) Wie viele "*" werden gezeigt?

```
' Verketten3
balken$ = "*"
FOR i = 1 TO 5
 balken$ = balken$ + balken$
NEXT
PRINT balken$
```

b) Wie viele "*" werden gezeigt, wenn die Zählvariable i bis 10 läuft?
Überprüfe deine Vermutung, indem du die Zeile

```
PRINT LEN(balken$)
```

hinzufügst.

11) Die Funktion LEN gibt die Länge des angegebenen Strings zurück. Experimentiere im Direktmodus:

```
PRINT LEN("abcd")
a$="Hallo"
PRINT LEN(a$)
```

Für viele Zweck müssen wir auf die einzelnen Zeichen eines Strings zugreifen bzw. einzelne Zeichen in einem String ändern. Hierzu stellt Q-BASIC die Funktion/Anweisung MID$ zu Verfügung.

12) a) Beobachte die Wirkung des folgenden Programms:

```
' Teilstring
CLS
text$ = "Vater"
laenge = LEN(text$)
FOR i = 1 TO laenge
  PRINT MID$(text$,i,1)
NEXT
```

b) Ersetze die 1 in der obigen MID$-Anweisung durch 2,3,4,... Was beobachtest du?

Also: Die Funktion **MID$(text$,i,k)** gibt vom String **text$** den Teilstring der Länge k ab dem i-ten Zeichen (einschließlich) zurück (falls dabei über das Ende des Strings in **text$** hinausgegangen wird, werden die nicht vorhandenen Zeichen ignoriert).

13) a) Welchen Ausdruck beobachtest du jetzt?

```
' Zeichen ersetzen
text$="Vater"
PRINT text$
MID$(text$,1,1)="K"
PRINT text$
```

b) Das Programm soll "Mutter" in "Butter" ("Schwester" in "Orchester") verwandeln.

Also: die Anweisung **MID$(text$,i,k) = a$** ersetzt im String **text$** die Teilzeichenkette der Länge k ab dem i-ten Zeichen durch a$.

14) Ein einzugebendes Wort soll "gespiegelt" d.h. in umgekehrter Reihenfolge seiner Buchstaben ausgegeben werden.

```
' Wortspiegeln
CLS
INPUT "Wort eingeben: ", wort$
laenge = LEN(wort$)
FOR i = laenge TO 1 STEP -1
 PRINT MID$(wort$,i,1);
NEXT
```

a) Teste das Programm mit den Eingaben: REGAL, LAGER, RENTNER, EIN NEGER MIT GAZELLE ZAGT IM REGEN NIE.
b) Das "Spiegelwort" soll in einer Variablen **invers$** gespeichert werden.

15) Schreibe ein Programm, das ein eingegebenes Wort daraufhin überprüft, ob es ein "Spiegelwort" (Palindrom) ist, d.h. von vorwärts wie rückwärts gelesen denselben Sinn ergibt.

16) Vervollständige das nachstehende Programm. Es soll in einem einzugebenden Text die Buchstaben "a" und "o" vertauschen. (Aus "Hase" wird also "Hose", aus "hallo" wird "holla".)

```
' a-o-Tausch
INPUT "Text eingeben: ", text$
l = LEN(text$)
FOR i=1 TO l
 IF MID$(text$,i,1)=     THEN
   MID$(text$,i,1)=
 ELSEIF MID$(text$,i,1)=     THEN
   MID$(text$,i,1)=
 END IF
NEXT
PRINT text$
```

5.2 Spiele

Zahlenmerken

Bei diesem Gedächtnisspiel wird kurz eine Folge zufällig gewählter Ziffern gezeigt, dann wird der Bildschirm wieder gelöscht. Der Spieler muss nun die Ziffernfolge wiedergeben. Nach jeder erfolgreichen Wiederholung wird die Folge um eine Ziffer länger ...

Der Algorithmus für dieses Spiel lässt sich so formulieren:

solange Eingabe = gezeigter Folge	
	Erzeuge zufällige Ziffer und hänge sie an die bisherige Folge an. Zeige die Ziffernfolge. Lösche den Schirm und nehme die Folge des Spielers entgegen.

Die Ziffernfolge speichern wir in der Textvariablen **folge$**. (In einer numerischen Variablen würde die Folge als Zahl aufgefasst und Zahlen mit vielen Stellen werden vom Computer in die Exponentialdarstellung umgewandelt.) In Zeile 4 wird die neu hinzukommende Ziffer zufällig erzeugt: Mit Hilfe der "Zufallszahlenfunktion" aus Kapitel 2 erzeugen wir eine Zufallszahl z zwischen 1 und 10 und wählen das z-te Zeichen aus dem String "12...90" der Ziffern. Diese Ziffer hängen wir an die in der Variablen **folge$** gespeicherte Folge an. Die neue Ziffernfolge wird jetzt 2 Sekunden lang gezeigt, dann wird der Bildschirm gelöscht und der Benutzer zur Wiedergabe der Folge aufgefordert. Dies wiederholt sich, bis sich die eingegebene Folge von der gezeigten unterscheidet.

```
' gedächtnisspiel
ziffern$="1234567890": eingabe$="": folge$=""
WHILE eingabe$=folge$
 z=zufallszahl(1,10): zeichen$=MID$(ziffern$,z,1)
 folge$=folge$+zeichen$
 CLS
 PRINT "Ziffernfolge jetzt: "; folge$
 SLEEP 2
 CLS
 INPUT "Gib die Folge wieder: ", eingabe$
WEND
PRINT
PRINT "Leider falsch. Die richtige Folge war: ";folge$
```

```
FUNCTION zufallszahl(n,m)
 RANDOMIZE TIMER
 zufallszahl = INT(RND*(m-n+1)+n)
END FUNCTION
```

17) Gib das Programm ein und probiere es aus. Bei welcher Länge der Ziffernfolge hast du einen Fehler gemacht?

18) Die Gedächtnisleistung, nämlich die Länge der Ziffernfolge beim Abbruch, soll bewertet und kommentiert werden.

Wortraten

Dieses Spiel legen wir als Partnerspiel an. Ein Spieler gibt, ohne dass sein Partner zusehen darf, ein Wort ein. Der andere Spieler soll dieses Wort raten. Anschließend tauschen die Spieler die Rollen. Der ratende Spieler kann im Wort vermutete Buchstaben nennen. Diese werden dann -falls vorhanden- an der richtigen Stelle angezeigt. (Alle Buchstaben werden klein geschrieben.)
Er kann auch das ganze Wort auf einmal nennen.
Ein möglicher Dialog:

Gesucht wird: - - - -
Lösungsbuchstabe oder -wort? e
Gesucht wird: e - - e
Lösungsbuchstabe oder -wort? erde
Richtig, gesucht war erde.

Der Algorithmus für das Spiel lässt sich so darstellen:

Eingabe Ratewort Schirm löschen Setze Hilfswort aus "-": Eingabe =""	
Solange Hilfswort<>Ratewort und Eingabe<> Ratewort	
	Zeige Hilfswort Eingabe: Lösungsbuchstabe- oder -Wort Wenn Buchstabe eingegeben diesen in Hilfswort einbauen

```
' Wortratespiel
CLS
INPUT "Bitte das Ratewort eingeben: ", ratewort$
CLS
laenge=LEN(wort$): eingabe$="": hilfswort$=""
FOR i=1 TO laenge
 hilfswort$=hilfswort$+"-"
NEXT
WHILE hilfswort$<>wort$ AND eingabe$<>ratewort$
 PRINT
 PRINT "Gesucht wird "; hilfswort$
 PRINT
 INPUT "Lösungsbuchstabe oder -wort: ", eingabe$
 IF LEN(eingabe$)=1 THEN buchstabeeinbauen
WEND
PRINT "Richtig, Lösungswort ist ";ratewort$
```

```
SUB buchstabeeinbauen
 SHARED laenge, ratewort$, hilfswort$, eingabe$
 FOR i=1 TO laenge
  IF MID$(ratewort$,i,1)=eingabe$ THEN
   MID$(hilfswort$,i,1)=eingabe$
  END IF
 NEXT
END SUB
```

Zum Programm: Nachdem die Länge des eingegebenen Ratewortes bestimmt ist, wird als Erstes ein Hilfswort aus "-" gebildet und ausgegeben. Nachdem der Spieler seinen Lösungsbuchstaben eingegeben hat, wird in der Prozedur **buchstabeeinbauen** das Ratewort Buchstabe für Buchstabe mit der Eingabe verglichen. Kommt der eingegebene Buchstabe an der Stelle i im Ratewort vor, so wird an derselben Stelle i der Buchstabe in das Hilfswort eingefügt. Die Prozedur benötigt vom Hauptprogramm die Variablen *laenge*, *ratewort$*, *hilfswort$* und *eingabe$*, diese Variablen müssen also zuerst für die Prozedur zugänglich gemacht werden mit der SHARED-Anweisung.

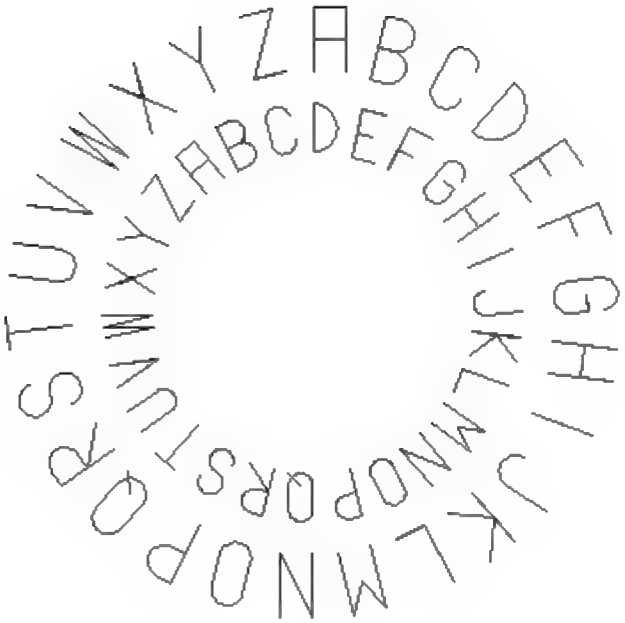
19) Ergänze das Programm so, dass die Anzahl der benötigten Rateversuche mit ausgegeben und kommentiert wird.

20) Es sollen höchstens 10 Rateversuche erlaubt sein.

5.3 Geheimschriften

Chiffrieren nach Cäsar

Schon lange wurden wichtige Nachrichten vor dem Lesen durch Unbefugte dadurch geschützt, dass man sie verschlüsselte. Julius Cäsar z.B. ersetzte jeden Buchstaben des Klartextes durch den Buchstaben, der im Alphabet drei Stellen hinter ihm kommt.
Welche Nachricht hat Cäsar im Bild geschrieben? (Der Klartext ist natürlich lateinisch, lass ihn dir von deinem Lehrer übersetzen.) Wir können Cäsars Verschlüsselungsmethode an zwei gegeneinander verdrehten Buchstabenscheiben darstellen:



Numerisch lässt sich diese zyklische Vertauschung so verwirklichen: Wir ordnen den 26 Buchstaben des Alphabets die Zahlen 0 bis 25 zu. Cäsars Chiffrierung bedeutet für die zugeordneten Zahlen den Übergang von n nach n+3, bzw. (n+3)-26, falls n+3>25 ist.
Wir wollen ein Programm schreiben, das einen eingegebenen Text nach der Methode von Cäsar verschlüsselt. Der Algorithmus hierfür ist jetzt klar:

Für i=1 bis Textlänge	
	Bestimme die Nummer n des i-ten Buchstabens n=n+3 falls n>25 dann n=n-26 n ist die Nummer des Codebuchstabens

Unser Problem: Wie finden wir die einem Buchstaben zugeordnete Nummer und wie umgekehrt den einer Zahl (zwischen 0 und 25) entsprechenden Buchstaben?
Hier ist ein kurzer Exkurs über die Darstellung von Zahlen und Zeichen im Rechner nützlich.

Der ASCII-Code

Ein Computer kann im Grunde nur das Vorhandensein oder das Fehlen von Strom unterscheiden. Das Vorhandensein von Strom wird symbolisiert durch eine 1, das Fehlen von Strom durch eine 0. Im Computer gibt es eine Vielzahl von Schaltungen, die solche Zustände realisieren. Ein solcher "Elementarspeicher" stellt also ein Bit (englisch: binary digit) d.h. eine Dualziffer 0 oder 1 dar. Je acht solcher Speicherelemente sind zusammengefasst zu einem Byte. Durch einen bestimmten elektromagnetischen Zustand eines solchen Speicherelements können somit die Zahlen von 0 (=00000000) bis 255 (mit der Darstellung 11111111 im Dualsystem) dargestellt werden.
Um nun Zeichen im Computer darzustellen, werden diese Zeichen durch Zahlen kodiert und dann die Codezahlen abgespeichert. Für die Codierung hat sich weitgehend der so genannte ASCII-Code durchgesetzt (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange): Dabei werden die Buchstaben a-z und A-Z jeweils durch 26 aufeinander folgende Zahlen kodiert. Insgesamt benutzt der ASCII-Code die Zahlen von 0 bis 127, kommt also mit 7 der 8 Bits in einem Byte aus. Das achte Bit wird für den sog. erweiterten ASCII-Code benützt.

21) Du kannst dir die Codierung der Zeichen mit nachstehendem Programm ansehen.

```
' ASCII-Code 1
INPUT "ASCII-Code von ", zeichen$
PRINT "ist "; ASC(zeichen$)
```

Welchen Code haben die Buchstaben a bis z (A bis Z), welchen die Ziffern 0 bis 9, welchen die Rechenzeichen +, -, *, /?

22) Dieses Programm zeigt umgekehrt zu einer Codezahl x (0<x<255) das zugehörige Zeichen.

```
' ASCII-Code 2
INPUT "Gib eine Zahl zwischen 1 und 255 ein: ", x
PRINT x; " ist die Codezahl von "; CHR$(x)
```

Gib die oben gefundenen Codezahlen ein.

Mit ASC() können wir den ASCII-Code eines Zeichens bestimmen, mit CHR$(x) können wir das Zeichen mit der Codezahl x bestimmen.

Nach diesem Exkurs können wir unser Problem lösen: Wir wollen den Buchstaben a bis z die Zahlen 0 bis 25 zuordnen. Dazu brauchen wir nur mittels **ASC** ihren ASCII-Code bestimmen und vom gefundenen Codewert die Zahl a=ASC("a") subtrahieren. Umgekehrt erhalten wir von einer Zahl n zwischen 0 und 25 den ASCII-Code des entsprechenden Buchstabens, indem wir a addieren; den Buchstaben selber erhalten wir dann mit der Funktion **CHR$**.
Das nachstehende Programm realisiert Cäsars Verschlüsselungsidee. Zunächst fordern wir die zu verschlüsselnde Nachricht und die Verschiebezahl v an. (Dabei lassen wir nur sinnvolle Werte von v zu, also ganze Zahlen zwischen 1 und 25.) Auf den Text in **nachricht$** wenden wir die Funktion **verschluesselt$(nachricht$,v)** an (wobei Cäsar den Wert v=3 gewählt hatte).

```
' Kodieren nach Cäsar
CLS
INPUT "Gib die zu verschlüsselnde Nachricht ein:
",nachricht$
v=0
WHILE v<1 OR v>25 OR v<>INT(v)
 INPUT "Verschiebezahl v =? (0<v<26) ", v
WEND
codiert$=verschluesselt$(nachricht$,v)
PRINT "Verschlüsselte Nachricht: ";codiert$

FUNCTION verschluesselt$(text$,v)
 a=ASC("a"): z=ASC("z"): laenge=LEN(text$)
 FOR i=1 TO laenge
  code=ASC(MID$(text$,i,1))
  IF code>=a AND code<=z THEN    'Zeichen ein Buchstabe?
    n=code-a  'zugeordnete Zahl zwischen 0 und 25
    n=n+v  'verschieben
    IF n>25 THEN n=n-26
    b$=CHR$(a+n)   'Verschlüsselter Buchstabe
    MID$(text$,i,1)=b$
  END IF
 NEXT
 verschluesselt$=text$
END FUNCTION
```

Da die Funktion einen String zurückgibt, muss ihr Name auf "$" enden. Die Funktion ersetzt in dem String, der in der Variablen **text$** gespeichert ist, jeden Buchstaben durch denjenigen Buchstaben, der im Alphabet v Stellen hinter ihm kommt (wobei diese Verschiebung als "Ringtausch" angelegt ist).

23) Verschlüssele folgende Nachrichten (wähle v=15):
angriff morgen frueh.
Dies ist eine streng geheime Botschaft!

24) a) Entschlüssele folgende Nachricht von Cäsar: dqjulii prujhq iuxhk
Du kannst die Funktion **verschluesselt$** auch zum Entschlüsseln benutzen. Welchen Wert musst du aber für v wählen? (Zum Verschlüsseln wurde v=3 gewählt.)
b) Entschlüssele die Chiffretexte, die du in der vorigen Aufgabe erhalten hast. Wieder kannst du die Funktion **verschluesselt$** dazu verwenden, welchen Wert musst du aber für v wählen?

25) Was geschieht bei unserer Codierung mit Leerzeichen, Satzzeichen, Großbuchstaben? (Anmerkung: Vermeide Kommata im Text, s. unten Abschnitt 5.4.) Ändere die Funktion **verschluesselt$** so ab, dass Großbuchstaben im Klartext durch Kleinbuchstaben nach der Kodiervorschrift verschlüsselt werden. (Bei v=3 soll also z.B. "A" durch "d" verschlüsselt werden.)

26) Der Chiffretext soll nur Kleinbuchstaben aber keine Leerzeichen und Satzzeichen enthalten.

27) Wir verwenden eine andere Chiffrierregel: Die Buchstaben sollen "gespiegelt" werden, d.h. "a" und "z" werden vertauscht, ebenso "b" und "y", "c" und "x" usw. Schreibe eine entsprechende Funktion **gespiegelt$** und kodiere die Texte von Aufgabe 23.
Wie entschlüsselst du einen nach dieser Regel chiffrierten Text?

Chiffrieren mit Schlüsseltexten

Der Mangel von Cäsars Chiffriermethode ist offensichtlich: Eine sorgfältige Häufigkeitsanalyse der vorkommenden Zeichen und Zeichenfolgen wird ausreichen, den chiffrierten Text zu entschlüsseln. Diese Schwäche weist folgende, bis heute weit verbreitete, Chiffriermethode nicht mehr auf: Wir "überlagern" den Klartext mit einem bestimmten Schlüsseltext, addieren also gliedweise zum Code der Buchstaben des Klartextes den Code der entsprechenden Buchstaben des Schlüsseltextes (und subtrahieren 26, falls die Summe größer ist als 25). Den Buchstaben, der diese Summe als Nummer trägt, wählen wir zur Chiffrierung.

Beispiel:

Klartext	a	n	g	r	i	f	f	m	o	r	g	e	n	f	r	u	e	h
Zahlencode	0	13	6	17	8	5	5	12	14	17	6	4	13	5	17	20	4	7
Schlüssel	e	i	n	e	s	t	a	g	e	s	s	a	g	t	e	d	i	e
Zahlencode	4	8	13	4	18	19	0	6	4	18	18	0	6	19	4	3	8	4
Chiffriert	e	v	t	v	a	y	f	s	s	j	y	e	t	y	v	x	m	l
Zahlencode	4	21	19	21	0	24	5	18	18	9	24	4	19	24	21	23	12	11

Die Codierung folgt also diesem Algorithmus:

laenge = Anzahl der Buchstaben des Klartextes	
Für i=1 bis laenge	
	Bestimme von den i-ten Buchstaben des Klartextes und des Schlüsseltextes die Nummern im Alphabet Addiere beide Zahlen Falls die Summe>25 ist, subtrahiere 26 Bestimme den Buchstaben mit dieser Nummer und hänge ihn an den bereits ermittelten Chiffretext an

Die Funktion **chiffriert$** folgt genau diesem Algorithmus. Um das Programm kurz und einfach zu halten, schreiben wir den Schlüsseltext in Kleinbuchstaben und lassen Leerzeichen und Satzzeichen weg. Ebenso setzen wir voraus, dass der Klartext in dieser Form eingegeben wird. Wir bestimmen von jedem Buchstaben des Klartextes und dem zugehörigen Buchstaben des Schlüsseltextes den ASCII-Code. Durch Subtrahieren von a=ORD("a") erhalten wir die Nummern der Buchstaben im Alphabet. Diese werden addiert und gegebenenfalls 26 subtrahiert. Indem wir zum Ergebnis wieder a addieren, erhalten wir den ASCII-Code des gesuchten Chiffrebuchstabens. Dieser ersetzt dann den ursprünglichen Buchstaben des Textes.

```
FUNCTION chiffriert$(text$)
 schluessel$="einestagessagtediemutterzurotkaeppchen"
 l=LEN(text$): a=ASC("a")
 FOR i=1 TO l
   code1=ASC(MID$(text$,i,1))-a
   code2=ASC(MID$(schluessel$,i,1))-a
   code=code1+code2
   IF code>25 THEN code=code-26
   asci=code+a
   zeichen$=CHR$(asci)
   MID$(text$,i,1)=zeichen$
 NEXT
 chiffriert$=text$
END FUNCTION
```

28) Schreibe noch ein Hauptprogramm, das die Eingabe des Klartextes anfordert und die Funktion **chiffriert$** aufruft. Überprüfe nun obiges Beispiel. Beachte: Bei der Eingabe des Klartextes darfst du nur Kleinbuchstaben verwenden und musst den Text ohne Leerzeichen und ohne Satzzeichen eingeben.

29) Schreibe eine entsprechende Prozedur zum Entschlüsseln.

30) Bei unserer Funktion **chiffriert$** musste der Klartext in Kleinbuchstaben und ohne Leerzeichen und Satzzeichen eingegeben werden. Schreibe eine Funktion **wandelt$**, die in einem Text Großbuchstaben durch die entsprechenden Kleinbuchstaben ersetzt und alle Zeichen, außer den Buchstaben, entfernt.

5.4 Textanalysen

"Ein Computer kann Unmengen von Informationen in kürzester Zeit verarbeiten. Oft würden Menschen für dieselbe Arbeit, die ein Computer in Minuten verrichtet, ein ganzes Leben brauchen. Aber auch dort, wo der Mensch ganz gut ohne Computer zurecht käme, wird dieser verwendet, weil er schneller, genauer, zuverlässiger arbeitet. Der Computer ist aber keine unheimliche, alles wissende Maschine. Er verarbeitet nur Daten, die ihm von Menschen gegeben wurden, nach Anweisungen, die sich Menschen ausgedacht haben."

Solche Texte findest du in Büchern oder Zeitungsberichten über den Computer. Wir wollen diesen Text benutzen, um an ihm verschiedene Operationen durchzuführen und um dabei zu lernen, wie Computer ihre Tätigkeit ausführen.

31) Zähle ab, wie oft in obigem Text das Wort "Computer" vorkommt. Zähle nun ab, wie oft die Zeichenfolge "en" im Text vorkommt.

Die Aufgabe, die Häufigkeit eines sinnvollen Wortes in einem Text zu bestimmen, ist dir sicher leicht gefallen, dagegen hattest du schon mehr Mühe, die Häufigkeit irgendeiner Zeichenfolge wie "en" im Text sicher zu bestimmen. Du hast dies auch als eine ziemlich sinnlose Aufgabe empfunden - aber was heißt "sinnlos"?
Die Zeichenfolge "en" ist in der deutschen Sprache ungewöhnlich häufig und es kann im Zusammenhang von Chiffrier- und Dechiffriermethoden, wie du sie im vorigen Abschnitt kennen gelernt hast, außerordentlich wichtig sein, solche Häufigkeitsanalysen durchzuführen. Unser Ziel ist es, ein Programm zu schreiben, das einen Text nach einem Suchwort hin durchmustert und die Häufigkeit des Suchworts im Text feststellt. Der Computer wird, mit diesem Programm versehen, schnell und zuverlässig seine Arbeit tun und keinen Unterschied zwischen "sinnvollen" und "sinnlosen" Suchwörtern machen. Es ist aber Aufgabe des Menschen, dieses Programm in einem sinnvollen Zusammenhang zu verwenden und mit den vom Computer ermittelten Ergebnissen vernünftig zu argumentieren.

Eingabe längerer Texte

Bevor der Computer einen Text analysieren kann, muss der Text zuerst eingegeben sein. Mit der **INPUT**-Anweisung können nur kürzere Texte (bis zu 255 Zeichen) von der Tastatur gelesen werden. Will man längere Texte in einer Variablen abspeichern, kann man hierzu das nachstehende Programm benutzen:

```
' Texteingabe
CLS
PRINT "Text zeilenweise eingeben.
PRINT "Texteingabe mit Leerzeile beenden."
zeile$="a": text$=""
WHILE zeile$<>""
 LINE INPUT zeile$
 text$=text$+CHR$(13)+zeile$
WEND
PRINT
PRINT "Du hast folgenden Text eingegeben:"
PRINT text$
```

Der **INPUT**-Befehl hat bei der Eingabe von Texten die unangenehme Eigenschaft, neben der EINGABE-Taste auch ein Komma als Ende der Eingabe zu lesen, sodass Texte mit Kommata mit **INPUT** nicht eingegeben werden können. Deswegen verwenden wir **LINE INPUT**, eine Anweisung die wie **INPUT** wirkt, aber Kommata akzeptiert. Nach jeder Zeile fügen wir ein RETURN (CHR$(13)) hinzu. Damit sorgen wir für einen Zeilenumbruch an der entsprechenden Stelle beim Ausdruck von **text$**. (Und außerdem für ein späteres korrektes Einlesen von der Datei auf Diskette mit **LINE INPUT.)** Der eingegebene Text wird in der Variablen **text$** gespeichert. Du beendest die Texteingabe durch Eingabe einer "Leerzeile" (also zweimal nacheinander die RETURN-Taste drücken). Zum Schluss wird der eingegebene Text gezeigt.

32) Teste das Programm mit der Eingabe eines längeren Textes.

Arbeit mit Dateien

Einen längeren Text in den Computer einzugeben, ist eine mühsame Arbeit. Wir wollen uns diese Arbeit nur einmal machen und den eingegebenen Text für spätere Zwecke auf Diskette speichern, von wo wir ihn dann mühelos bei Bedarf wieder einladen können. Unser Text wird in einer sog. "Datei" (englisch **file**) auf der Diskette gespeichert. Diese Datei muss einen Namen erhalten, damit wir sie später auf der Diskette wieder finden.

```
' Langen Text speichern
CLS
PRINT "Bitte Dateinamen eingeben (evtl. mit Laufwerk und Pfad):"
INPUT name$
PRINT "Zu speichernden Text zeilenweise eingeben."
PRINT "Texteingabe mit Leerzeile beenden."
zeile$ = "a": text$ = ""
WHILE zeile$<>""
 LINE INPUT zeile$
 text$ = text$ + CHR$(13) + zeile$
WEND
speichern name$, text$

SUB speichern(dateiname$,a$)
 OPEN dateiname$ FOR OUTPUT AS #1
 PRINT #1, a$
 CLOSE #1
END SUB
```

33) Starte das Programm, gib den Dateinamen ein (Vorschlag: a:computer.dat) und dann obigen Text über Computer. Im Laufwerk A: muss eine Diskette liegen. Sobald du die Texteingabe beendet hast, bemerkst du, wie das Diskettenlaufwerk kurz arbeitet. Im Inhaltsverzeichnis der Diskette im Laufwerk A: findest du die Datei "computer.dat".

Die Prozedur **speichern** öffnet auf der Diskette eine Datei mit dem Namen der in der Variablen **dateiname$** steht und gibt der eben geöffneten Datei die Dateinummmer 1. Außerdem teilen wir dem System mit, dass in diese Datei geschrieben werden soll (sie soll den PRINT-Output aufnehmen). Die Dateinummer dient zur Identifizierung der Datei im laufenden Programm (es können nämlich mehrere Dateien gleichzeitig geöffnet sein), bei späteren Schreib- und Lesebefehlen muss die Nummer der jeweils gewünschten Datei angegeben werden. (Dagegen dient der Dateiname zur Identifizierung der Datei auf der Diskette.) Die zweite Zeile der Prozedur ist ein solcher Schreibbefehl: In die Datei mit der Nummer 1 soll der Inhalt der Variablen **a$** geschrieben werden. Jede geöffnete Datei muss wieder geschlossen werden, dazu dient die letzte Zeile. Beim Aufruf der Prozedur **speichern** bekommen die Variablen **dateiname$ und a$** ihren Wert übertragen. Der Aufruf geschieht einfach durch Nennen des Namens der Prozedur, gefolgt von der Liste der zu übertragenden Variablen (anders als beim Funktionsaufruf, ist diese Liste nicht in Klammern zu setzen).

34) Lösche das Hauptmodul und gib dieses kleine Programm ein.

```
PRINT "Text eingeben:"
INPUT text$
speichern "a:test.dat", text$
```

Starte nun das Programm und gib einen kleinen Text ein.

Du hast jetzt auf der Diskette zwei Textdateien: die Datei "computer.dat" von Aufgabe 33 und die Datei "test.dat" von Aufgabe 34. Natürlich wollen wir nun wissen, wie wir diese Texte wieder von der Diskette in den Arbeitsspeicher des Computers laden können.

35) Gib nachstehendes Programm ein. Lies den Text der Dateien "test.dat" und "computer.dat" ein.

```
' text laden
CLS
PRINT "Namen der Datei, die gelesen werden soll, eingeben:"
INPUT name$
einlesen name$
```

```
PRINT "In der Datei ";name$;" ist folgender Text:"
PRINT text$

SUB einlesen(dateiname$)
 SHARED text$
 OPEN dateiname$ FOR INPUT AS #1
 WHILE NOT EOF(1)
  LINE INPUT #1, zeile$
  text$=text$+CHR$(13)+zeile$
 WEND
 CLOSE #1
END SUB
```

Der Text wird in die Variable **text$** eingelesen, die dem Hauptprogramm zur Verfügung stehen soll. Daher muss die Variable **text$** mittels SHARED dem Hauptprogramm bekannt gemacht werden. Es wird eine Datei geöffnet unter Angabe von Dateiname und Dateinummer, jetzt allerdings zum Lesen (mittels INPUT bzw. LINE INPUT). Das Einlesen von einer Datei ist genauso zu handhaben wie von der Tastatur und vollzieht sich mittels LINE INPUT: Es wird jeweils bis zum nächsten RETURN-Zeichen (CHR$(13)) gelesen, der Text zu dem bereits gelesen hinzugefügt. Das Ende des Einlesevorgangs ist erreicht, wenn das von Q-BASIC an den Schluss der Datei gesetzte End-of-File-Zeichen gelesen wird, daher die Bedingung WHILE NOT EOF(1).

Suchen

Unser Ziel ist eine Funktion **gesucht**, die einen Text nach einem Suchwort durchmustert und seine Häufigkeit im Text zurückgibt. Wir denken uns dazu eine Schablone, die den ganzen Text verdeckt und nur in einem Sichtfenster ein Teilstück des Textes von der Länge des Suchwortes frei lässt. Dieses Sichtfenster schieben wir vom Anfang des Textes bis zum Ende durch und notieren jeden Fall, wo im Sichtfenster gerade das Suchwort auftaucht. Genau auf diese Weise arbeitet der folgende Algorithmus:

Setze den Zähler n auf 0.
Bestimme die Länge s des Suchworts.
Bilde vom Anfang des Textes bis zum Ende alle Teilstrings der Länge s.
Falls ein Teilstring mit dem Suchwort übereinstimmt, erhöhe den Zähler n um 1.

Wir haben in Q-BASIC schon mehrmals Teilstrings gebildet. Zur Übung wiederholen wir:

36) Gib nachstehendes Programm ein und beobachte seine Wirkung.

```
' Teilstrings
text$="Ein Computer verarbeitet Daten."
a$=MID$(text$,1,3): b$=MID$(text$,17,8)
PRINT a$
PRINT b$
```

Du siehst: Mit MID$(text$,i,k) wird von der Zeichenkette, die in der Variablen **text$** gespeichert ist, der Teilstring der Länge k ab dem i-ten Zeichen herausgegriffen.

Die Funktion **gesucht(text$,suchwort$)** gibt die Häufigkeit des Vorkommens des Strings **suchwort$** im String **text$** zurück. Die Funktion bildet alle Teilstrings des Textes, welche die Länge s des Suchworts haben und vergleicht sie mit diesem. In der Variablen n wird das Vorkommen des Suchworts im Text gezählt. Der erste Teilstring geht vom ersten bis zum s-ten Textzeichen, der letzte endet beim l-ten Textzeichen, beginnt also beim (l-s+1)-ten Zeichen, wobei l die Länge von **text$** ist.

```
FUNCTION gesucht(text$,suchwort$)
 n=0: s=LEN(suchwort$): l=LEN(text$)
 FOR i=1 TO l-s+1
   IF MID$(text$,i,s)=suchwort$ THEN n=n+1
 NEXT
 gesucht=n
END FUNCTION
```

37) Das Hauptprogramm für unseren Suchalgorithmus lautet:

```
' Teilstrings in Texten suchen
CLS
INPUT "Dateiname (Laufwerk/Pfad): ", name$
INPUT "Suchwort: ", suchwort$
einlesen name$
n=gesucht(text$,suchwort$)
PRINT suchwort$;" kommt im Text ";n;"-mal vor."
```

a) Gib das Programm ein. Ergänze es um die Prozedur **einlesen**.
b)Starte das Programm und bestimme die Häufigkeit der Zeichenfolgen "Computer", "Mensch", "en" "er", "e", "a".

Vokale und Konsonanten

COLPO IN DOPPIOPETTO

LONDRA - Era stato il "deposito piu sicuro del mondo". Ma domenica pomeriggio, con una sequenza da grande cinema, due distinti signori in fumo di Londra, aiutati da alcuni complici, hanno alleggerito il contentuto delle sue cassette di sicurezza di almeno 20 miliardi di lire. E'successo di fronte ai famosi grandi magazzini Harrods. I rapinatori si sono presentati come potenziali clienti e, una volta nei sotteranei, hanno bloccato il direttore e due guardie agendo indisturbati.

MUDSLIDE KILLS 22 AT FRENCH CAMPSITE

ANNECY, France - Fifty persons were killed or reported missing after an earthen dam collapsed and a wave of mud and water from a rain-swollen river swept into a campground filled with vacationers near here, authorities said Wednesday. Hundred of rescue workers were digging through dirt and rocks in the French Alpine village of Le Grand Bornand. By midday, the death toll reached 22, with 28 others feared dead. Ten persons were seriously injured and dozens were treated for shock.

Dies sind zwei kurze Texte aus einer italienischen bzw. einer englischen Zeitung. Auf den ersten Blick fällt die größere Häufigkeit von Vokalen im italienischen Text auf. Ist das Verhältnis "Zahl der Vokale zu Zahl der Konsonanten" sprachspezifisch? Bei einer Untersuchung dieser Frage kann die nachstehende Funktion **vok(text$)** sehr nützlich sein. Sie gibt die Anzahl der Vokale im Text **text$** zurück.
Der zugrunde liegende Algorithmus ist einfach:

laenge = LEN(text$)	
Für i=1 bis laenge	
	Wenn das i-te Textzeichen ein Vokal ist, dann zähle die Anzahl der Vokale um 1 weiter

Die Überprüfung, ob ein Textzeichen ein Vokal ist, wird mit Hilfe der Funktion **INSTR** durchgeführt. **INSTR(text$,teil$)** gibt die Position an, ab der **teil$** als Teilstring in **text$** vorkommt (bzw. 0, falls **teil$** kein Teilstring von **text$** ist).

38) Teste im Direktmodus

```
PRINT INSTR("aeiou,"e")
PRINT INSTR("Arbeitsbuch Q-BASIC","BASIC")
PRINT INSTR("AEIOU","e")
```

39) a) Speichere obige Texte auf der Diskette unter dem Namen "ital.dat" bzw. "engl.dat".
b) Schreibe ein Programm, das Texte von Diskette einliest und nachstehende Funktion **vok** aufruft.

```
FUNCTION vok(text$)
 l=LEN(text$): v=0
 vokale$="aeiouyAEIOUY"
 FOR i=1 TO l
   a$=MID$(text$,i,1)
   IF INSTR(vokale$,a$)>0 THEN v=v+1
 NEXT i
 vok=v
END FUNCTION
```

40) Schreibe eine entsprechende Funktion **kon(text$)** zur Bestimmung der Anzahl der Konsonanten. Bestimme nun das Verhältnis Konsonanten:Vokale in den drei Texten.

Wörter

Sind die Wörter in einem Zeitungsartikel im Durchschnitt kürzer als in einem literarischen Werk? Ist die mittlere Wortlänge bei Texten verschiedener Sprachen deutlich verschieden? Bei einer Untersuchung solcher Fragen kann die nachstehende Prozedur **wort(text$)** sehr nützlich sein. Sie bestimmt die Anzahl der Wörter im Text **text$** und die mittlere Wortlänge.

Wörter sind lückenlose Folgen von Buchstaben. Sie werden getrennt durch Leerzeichen oder Satzzeichen. Ob das i-te Zeichen a$ im Text ein Buchstabe ist, testen wir wieder mit der INSTR-Funktion. Solange nur Buchstaben aufeinander folgen, können wir die Wortlänge weiterzählen (Zeile 7,8), andernfalls ist ein Wort zu Ende und wir addieren die ermittelte Wortlänge zur bisherigen Summe aller Wortlängen, zählen die Wortzahl um 1 weiter und setzen die Variable **wortlaenge** für das nächste Wort wieder auf Null. Wir müssen aber aufpassen. Folgen mehrere "Nichtbuchstaben" aufeinander (z.B. Satzzeichen und Leerzeichen), so dürfen wir natürlich die Wortzahl nicht jedesmal erhöhen, daher in Zeile 10 zuerst die Kontrolle, ob überhaupt schon ein Wort begonnen ist, das durch einen "Nichtbuchstaben" beendet wird. Das letzte Wort im Text wird bei diesem Algorithmus nur gezählt, wenn der Text mit einem "Nichtbuchstaben" abschließt. Daher fügen wir in Zeile 2 vorsichtshalber ein Leerzeichen am Schluß des Textes an.

```
SUB wort(text$)
 text$=text$+" "
 b$="abcdefghijklmnopqrstuvwxyzäöüßABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ"
 zahl=0: wortlaenge=0: summe=0: l=LEN(text$)
 FOR i=1 TO l
  a$ = MID$(text$,i,1)
  IF INSTR(b$,a$)>0 THEN
    wortlaenge=wortlaenge+1
  ELSE
   IF wortlaenge>0 THEN summe=summe+wortlaenge: zahl=zahl+1:
   wortlaenge=0
  END IF
 NEXT
 PRINT "Der Text enthält "; zahl; " Wörter"
 PRINT "Mittlere Wortlänge: "; summe/zahl
END SUB
```

41) Schreibe ein Hauptprogramm, das einen Text von der Diskette einliest und die Prozedur **wort** aufruft. Der Name der einzulesenden Datei soll vom Benutzer erfragt werden.

42) Häufig verwendet wird die Funktion **INSTR**, um eine gegebene Antwort daraufhin zu untersuchen, ob sie Teil einer zulässigen Antwortzeichenkette ist:

```
antw$="a"
WHILE INSTR("1234",antw$)=0
  INPUT "Bitte eine der Ziffern 1,2,3,4 eingeben: ",antw$
WEND
PRINT "Du hast ";antw$;" eingegeben"
```

Was geschieht, wenn du einen Buchstaben eingibst, was wenn du die Zahl 7 eingibst? Wie reagiert das Programm auf die Eingabe der Zahl 11, wie auf Eingabe von 12? Bei welchen Eingaben wird also die WHILE-Schleife verlassen?

43) Setze die Prozeduren bzw. Funktionen **suchen**, **vok, kon** und **wort** zu einem Menü-Programm "Textanalyse" zusammen. Die zu analysierenden Texte seien bereits auf Diskette gespeichert.

```
        Welcher Text soll untersucht werden? ital.dat

        Textanalyse

1       Häufigkeit eines Zeichens oder einer Zeichenkette
2       Vokale und Konsonanten.
3       Wortzahl und mittlere Wortlänge.
4       Ende

Bitte wählen:
```

Du brauchst nur das Rahmenprogramm und die Menüprozedur neu zu schreiben.
Hinweis: Du hast alle wesentlichen Prozeduren und Funktionen bereits in früheren Aufgaben geschrieben. Q-BASIC erlaubt leider nicht das Hinzuladen alter Programme, du kannst dich aber so behelfen: Lade die Programme in ein Textverarbeitungssystem, führe sie dort zusammen und speichere das fertige Programm dann unformatiert ab. Du kannst es dann mit *Datei - Öffnen* nach Q-BASIC laden.

5.5 Aufgaben

44) Schreibe ein Trainingsprogramm zum Kleinen Einmaleins. Dialog:

```
Aufgaben zum Kleinen Einmaleins

 3*7 = ? 21  richtig

Noch eine Aufgabe?(j/n) j

8*6 = ? 42  falsch, 8*6 = 48

Noch eine Aufgabe?(j/n) n

Du hast 2 Aufgaben bearbeitet und dabei 1 Fehler gemacht.
```

45) a) Es soll ein 'Zufallswort' erzeugt werden. Die Länge des Zufallsworts liegt zufällig zwischen 2 und 8.
b) Im 'Zufallswort' soll sich kein Zeichen wiederholen.

46) Sprachwissenschaftler behaupten, dass ein Text in deutscher Sprache, aus dem alle Vokale entfernt wurden, immer noch verständlich sei. Schreibe ein Programm, welches aus einem eingegebenen Text alle Vokale entfernt und die übrigen Zeichen entsprechend zusammenrückt. Kannst du den Sprachwissenschaftlern zustimmen?

47) In einem Text sollen bestimmte Wörter in Großbuchstaben umgeschrieben werden. Schreibe eine entsprechende Funktion **gross$(wort$)**.
Beispiel: Im "Computertext" von Abschnitt 5.4 soll das Wort "Computer" in Großbuchstaben geschrieben werden.

48) In einem Text sollen bestimmte Wörter gesperrt gedruckt werden. Schreibe eine entsprechende Funktion **gesperrt$(wort$)**.

49) "Superhirn": Der Computer denkt sich eine vierstellige Zahl aus lauter verschiedenen Ziffern. Der Spieler soll sie erraten. Er gibt eine vierstellige Zahl ein und erhält anschließend folgende Informationen: Ist eine Ziffer erraten und an der richtigen Stelle, heißt es "Volltreffer", stimmt die Ziffer aber nicht die Stelle, heißt es "Halbtreffer".
Hinweis: Speichere die Ratezahl und die Eingabe in Textvariablen und verwende die Operationen, die du jetzt kennengelernt hast.

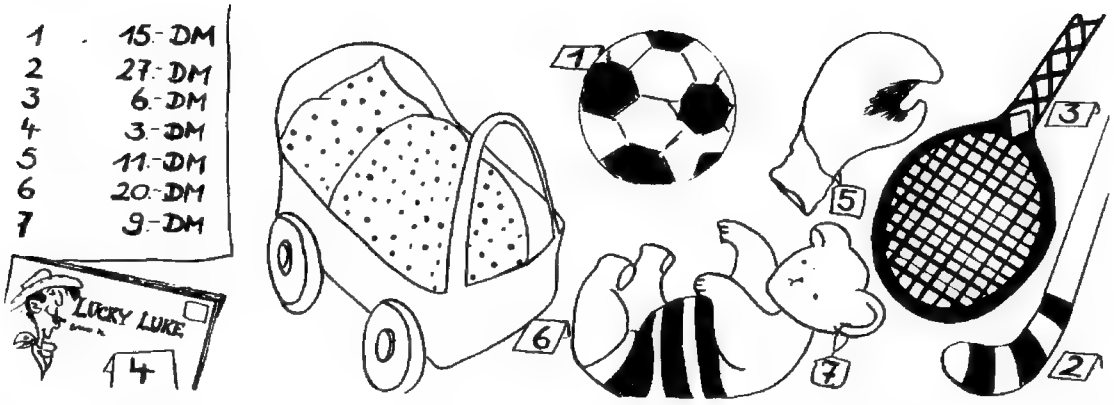
50) Eine einfache aber wirkungsvolle Chiffriermethode besteht darin, die Zeichen des Klartextes in ihrer Reihenfolge zu vertauschen. Das kann so geschehen: Wir schreiben den Text zeilenweise in eine Matrix und lesen den Chiffretext spaltenweise. Freie Plätze am Ende füllen wir mit 'x' auf.
Beispiel: Klartext: angriff morgen frueh.
Chiffretext: afeen nhgm xrofxirrxfguh

1	2	3	4	5	6
a	n	g	r	i	f
f		m	o	r	g
e	n		f	r	u
e	h	x	x	x	x

Der Schlüssel ist hier die Spaltenzahl 6. Zusammen mit der Textlänge ist dadurch die Zeilenzahl festgelegt. Der Adressat trägt den Ciffretext spaltenweise in die Matrix ein und liest sie zeilenweise. Für ein Verschlüsselungsprogramm nach dieser Methode numerieren wir die Zeichen des Klartextes durch. Dann drucken wir die Zeichen in dieser Anordnung:
zuerst die Zeichen mit den Nummern 1, 7, 13, 19,...
dann die Zeichen mit den Nummern 2, 8, 14, 20,...
dann die Zeichen mit den Nummern 3, 9, 15, 21,... usw.
a) Schreibe ein entsprechendes Codierprogramm und ein Decodierprogramm.
b) Schreibe Codier- und Decodierprogramm mit frei wählbarem Schlüssel s.

6 Felder und Dateien

6.1 Felder



Frederikes Klasse will beim Schulfest einen Flohmarkt veranstalten. Frederike nummeriert die Sachen und schreibt sich auf einer Liste die Preisvorschläge auf. Eine solche Liste kann man auch in einem Q-BASIC-Programm ablegen.

```
' Preisliste
DIM preis(20)

preis(1)=15
preis(2)=27
preis(3)=6
```

Die verschiedenen Preise werden alle unter dem gemeinsamen Namen **preis** abgespeichert, die Unterscheidung erfolgt über den *Index*, der in Klammern hinter dem Namen steht. In der *Dimensionierungsanweisung*

```
DIM preis(20)
```

wird im Speicher Platz für die 20 Variablen **preis(1), preis(2), preis(3), ..., preis(20)** geschaffen. In diese Variablen wird durch die Dimensionierung zunächst die Zahl 0 gespeichert. Die Gesamtheit dieser Variablen heißt ein *Feld* oder auch ein *array*.

1) Übertrage die restlichen Zahlen aus Frederikes Preisliste in das Q-BASIC-Programm und starte das Programm. Da bis jetzt keine Ausgabeanweisung enthalten ist, geschieht natürlich äußerlich nichts. Im Direktmodus kannst du aber nachprüfen, was der Computer jetzt "weiß":

```
PRINT preis(3)
PRINT preis(20)
PRINT preis(21)
```

Ergänze das Programm durch einen Ausgabeteil, der eine Preisliste in Tabellenform erzeugt. Verwende dazu eine FOR-Schleife und **PRINT USING**. Versuche die Eigenschaften von PRINT USING aus der Hilfe zu erfahren.

2) Da Frederike nicht weiß, wie viele Artikel sie am Schluss auf der Liste haben wird, möchte sie den Ausgabeteil so abändern, dass dafür die Anzahl der Artikel keine Rolle mehr spielt. Sie arbeitet mit einer **WHILE**-Schleife, die durchlaufen wird, solange der Preis größer als 0 ist. Falls nun ein neuer Artikel dazu kommt, braucht sie nur eine Zeile im Programm zu ergänzen, ohne am übrigen Programm etwas abzuändern.

Felder von Wörtern

Frederike will auch die Bezeichnung der Artikel auf der Liste notieren. Dazu braucht sie ein Feld für 20 Namen. Dann lautet die *Dimensionierungsanweisung*

```
DIM artbezeichnung$(20)
```

Danach werden die Artikelbezeichnungen abgelegt

```
artbezeichnung$(1)= "Fußball"
artbezeichnung$(2)= "Hockeyschläger"
artbezeichnung$(3)= "Family-Tennis"
```

3) Vervollständige diese Liste und schreibe Frederikes Programm so um, dass es Nr., Name, Preis in Tabellenform ausgibt. Verwende **PRINT USING**.

Simulationen (kann übersprungen werden)

4) Anton, Berta und Clemens spielen ein Würfelspiel. Zwei Würfel werden geworfen, ist die Augensumme höchstens 5, so gewinnt Anton, bei 6, 7, 8 gewinnt Berta, bei einer Augensumme von mindestens 9 gewinnt Clemens. Berta gewinnt laufend. Clemens denkt nach: "Die Augensumme 7 ist viel wahrscheinlicher als die Augensumme 2". Da Berta das nicht einsehen will, simuliert Clemens das Spiel 1000-mal auf dem Computer und zählt die Häufigkeiten der Augensummen 2, 3, 4, ... 12. Um diesen Häufigkeiten nicht 11 verschiedene Namen geben zu müssen, vereinbart er ein Feld:

```
DIM haeufigkeit(12)
```

In der Variablen **haeufigkeit(2)** zählt er, wie oft die Augensumme 2 auftritt usw.

```
'Würfelspiel
RANDOMIZE TIMER
DIM haeufigkeit(12)
FOR i=1 TO 1000
  x= INT(6*RND)+1
  y= INT(6*RND)+1
  summe=x+y
  haeufigkeit(summe)=haeufigkeit(summe)+1
NEXT i

FOR s=2 TO 12
  PRINT"Augensumme";s;"fiel";haeufigkeit(s);"mal."
NEXT s
```

Nach mehrmaligem Laufen des Programms verlangen Anton und Clemens eine Änderung der Spielregeln.

5) Lasse ein Balkendiagramm zu der Häufigkeitstabelle zeichnen.

6) Welche Augensummen kommen beim Spiel mit 3 Würfeln am häufigsten vor? Simuliere das Werfen dreier Würfel.

7) Immer wenn Mathematiklehrer Matke in eine neue Klasse kommt, wettet er, dass mindestens zwei Schüler am gleichen Tag Geburtstag haben. Wie groß ist die Wahrscheinlichkeit, dass er gewinnt, wenn die Klasse 30 Schüler hat ?

   Um dies näherungsweise beantworten zu können, wollen wir viele Klassen auf dem Computer simulieren. Wir beginnen zunächst mit einer Klasse. Für den Geburtstag eines Schülers gibt es 365 Möglichkeiten. Wir legen deshalb ein Feld **geburtstage** mit 365 Elementen an. Dabei soll **geburtstage(n)** angeben, wie viele Geburtstage am n-ten Tag des Jahres in der Klasse gefeiert werden. Die Geburtstage simulieren wir durch Zufallszahlen.

```
' Geburtstage
RANDOMIZE TIMER
DIM geburtstage(365)
FOR i=1 TO 30
  tag= INT(365*RND)+1
  geburtstage(tag)=geburtstage(tag)+1
NEXT i

max=0
FOR t=1 TO 365
  IF geburtstage(t)>max THEN max=geburtstage(t)
NEXT t
IF max>1 THEN PRINT"Matke hat gewonnen."
```

   Simuliere nun diesen Vorgang sehr oft und zähle, wie oft Matke gewinnt. Beantworte damit die oben gestellte Frage.

Sieb des Eratosthenes (kann übersprungen werden)

Das Verfahren des Erathostenes erzeugt eine Liste von Primzahlen. Es beruht auf folgendem Gedanken: Suchen wir alle Primzahlen bis zu einer gewissen Obergrenze (z.B. 100), so schreiben wir uns zunächst alle Zahlen von 2 bis zur Obergrenze auf.

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, . . . 100

Aus dieser Liste müssen alle Zahlen gestrichen werden, die echte Vielfache von kleineren Zahlen der Liste sind.
Wir streichen zunächst die Vielfachen von 2:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, . . . 100

Dann streichen wir zusätzlich die Vielfachen von 3:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, . . . 100

usw.

Die Zahlen die am Schluss übrig bleiben, sind Primzahlen.
Im Programm notieren wir in einem Feld **durchgestrichen**, welche Zahlen durchgestrichen sind. Dabei soll

durchgestrichen(i)=1 bedeuten, dass die Zahl i durchgestrichen,
durchgestrichen(i)=0 , dass i noch nicht durchgestrichen ist.

```
' Sieb des Eratosthenes
DIM SHARED n AS INTEGER
INPUT "Obergrenze ? ", n
DIM SHARED durchgestrichen(n) AS INTEGER
FOR i=2 TO n
  streicheVielfache i
NEXT i

SUB streicheVielfache(i)
  FOR k=2 TO n/i
   durchgestrichen(k*i)=1
  NEXT k
END SUB
```

8) Schreibe den Ausgabeteil des Programms und lasse dir die Primzahlen zwischen 1 und 100 ausdrucken.

9) Einige mathematische Überlegungen machen das Programm wesentlich schneller. Sicher ist dir vorhin schon aufgefallen, dass man die Vielfachen von 4 nicht mehr streichen muss, da diese schon als Vielfache von 2 gestrichen sind. Bei jeder Zahl, die schon gestrichen ist, sind auch die Vielfachen schon gestrichen. Ändere Zeile 5 ab zu

```
IF durchgestrichen(i)=0 THEN streicheVielfache i
```

Kommt man bei dem Verfahren zu einer Primzahl i, so ist 2*i schon als 2er-Zahl durchgestrichen, 3*i schon als 3er-Zahl usw. Es genügt also, bei i*i mit Durchstreichen zu beginnen. Ändere Zeile 8 entsprechend ab. Dieselbe Überlegung zeigt, dass es genügt, die **FOR**-Schleife in Zeile 4 nur bis **sqr(n)** laufen zu lassen. Ändere auch Zeile 4 entsprechend ab. Mit dem schnelleren Programm kannst du die Obergrenze größer wählen.

Gehirn-Training (kann übersprungen werden)

10) Tills Oma beschäftigt sich am liebsten mit der Rätselecke der Zeitung. Till schreibt ihr zum Geburtstag ein Knobelprogramm, das Wörter versteckt: Ein zufällig ausgewähltes Wort wird gespiegelt (d.h. in umgekehrter Buchstabenfolge angeordnet) und vorne und hinten werden noch zufällig gewählte Buchstaben angehängt, deren Anzahl natürlich auch zufällig sein muss. Zum Beispiel ist in "txamobez" das Wort "oma" versteckt.

Till geht so vor: Zunächst stellt er in einem Feld den Wortschatz zusammen, aus dem der Computer ein Wort zufällig wählt. Dieses wird gespiegelt, dann werden Buchstaben vorne und hinten angehängt. Der Computer schreibt die Buchstabenfolge auf den Bildschirm und fragt nach der Lösung, die Antwort wird bewertet.

```
' Wortverstecken
DIM SHARED wort$(40), n
DIM SHARED suchwort$
wortschatz
auswahl
spiegeln
anhängen
schreiben
fragen
```

```
SUB wortschatz
  n=3 'Zahl der Worte
  wort$(1)="oma"
  wort$(2)="geburtstag"
  wort$(3)="computer"
END SUB

SUB auswahl
  z=INT(n*RND)+1
  suchwort$=wort$(z)
END SUB
```

Till hat zunächst nur drei Wörter bereitgestellt, sich aber die Möglichkeit offengehalten, bis zu 40 Wörter einzugeben. Erweitere den Wortschatz und ergänze die fehlenden Prozeduren. Damit nicht immer dieselbe Folge kommt, muss noch an den Anfang RANDOMIZE TIMER.

11) Nicht nur in der Rätselecke der Zeitung, auch mit mathematischen Problemen kann man sein Gehirn trainieren. Hier ist ein Ein-Personen-Spiel zum Teilen von Zahlen:
Der Spieler wählt eine beliebige Obergrenze, z.B. 10. Aus der Reihe
1 2 3 4 5 6 7 8 9 10
darf er sich nun irgendeine Zahl ungleich 1 wählen. Der Computer bekommt alle Teiler der Zahl, die sich in der Reihe befinden. Nun holt sich wieder der Spieler eine Zahl aus der Restliste, wobei allerdings nur solche Zahlen erlaubt sind, die mindestens noch einen Teiler in der Liste haben. Der Computer erhält wieder alle Teiler aus der Liste. Das Spiel ist beendet, wenn der Spieler keinen Zug mehr machen kann. Falls noch Zahlen in der Liste sind, erhält diese der Computer. Spieler und Computer addieren jeweils ihre Zahlen; Sieger ist, wer die größere Summe erreicht. Hier siehst du zwei Spielabläufe für die Obergrenze 10.

```
              Reihe                Spieler wählt   Computer erhält

1  2  3  4  5  6  7  8  9  10          10              1, 2, 5
      3  4     6  7  8  9              9              3
         4     6  7  8                 8              4
               6  7              kein Zug möglich      6, 7

                          Summe:      27              28
Der Computer hat gewonnen.


1  2  3  4  5  6  7  8  9  10           7              1
   2  3  4  5  6     8  9  10           9              3
   2     4  5  6     8     10           6              2
         4  5        8     10          10              5
         4           8                  8              4

                          Summe:      40              15
Der Spieler hat gewonnen.
```

Aus der obigen Beschreibung ergibt sich das Hauptprogramm:

```
' Teilerspiel
DIM SHARED imSpiel(100)
DIM SHARED abbruch, obergrenze
DIM SHARED spieler, computer
Anleitung
ObergrenzeWaehlen
abbruch=0
WHILE abbruch=0
 ZahlenZeigen
 ZahlWaehlen
 AbbruchTesten
WEND
Auswertung
```

Wir verwenden ein Feld **imSpiel** , um uns zu merken, welche Zahlen noch im Spiel sind. Es soll
imSpiel(i)=1 bedeuten, dass die Zahl i noch im Spiel ist,
imSpiel(i)=0 , dass i nicht mehr im Spiel ist.

Hier sind Vorschläge für einige der benötigten Prozeduren.

```
SUB obergrenzeWaehlen
  PRINT "Welche Obergrenze wählst du ?"
  PRINT "Gib eine Zahl zwischen 5 und 100 ein !"
  INPUT obergrenze
  FOR i=1 TO obergrenze : imSpiel(i)=1 : NEXT
  spieler=0: computer=0
END SUB

SUB zahlWaehlen
  DO
    INPUT "Welche Zahl wählst du ? ", zahl
    IF imSpiel(zahl)=0 THEN
      PRINT zahl;"ist nicht mehr im Spiel"
      erlaubt=0
    ELSEIF computeranteil(zahl)=0 THEN
      PRINT zahl;"hat keinen Teiler in der Liste"
      erlaubt=0
    ELSE
      erlaubt=1
    ENDIF
  LOOP UNTIL erlaubt=1
  spieler=spieler+zahl
  computer=computer+computeranteil(zahl)
  FOR i=1 TO zahl
    IF zahl MOD i=0 THEN imSpiel(i)=0
  NEXT i
END SUB

SUB abbruchTesten
  i=1
  abbruch=1
  DO
    IF imSpiel(i)=1 THEN
      IF computeranteil(i)>0 THEN abbruch=0
    ENDIF
    i=i+1
  LOOP UNTIL abbruch=0 OR i>obergrenze
  IF abbruch=1 THEN
   PRINT "Spiel beendet."
   FOR i=1 TO obergrenze
    computer=computer+i*imSpiel(i)
   NEXT i
  END IF
END SUB

FUNCTION computeranteil(i)
  anteil=0
  FOR t=1 TO i/2
    IF i MOD t = 0 THEN anteil=anteil+imSpiel(t)*t
  NEXT t
  computeranteil = anteil
END FUNCTION
```

Schreibe die fehlenden Prozeduren.

Ordnen von Feldern

Olga hat sich in der Informatik-AG einige Begriffe gesondert aufgeschrieben und will daraus ein Stichwortverzeichnis machen. Dazu müssen diese Wörter alphabetisch geordnet werden.

Sie geht die ungeordnete Liste durch und sucht das Element, das auf Platz 1 gehört. Hat sie dieses gefunden, setzt sie es auf die neue Liste und streicht es auf der alten Liste durch. In der Restliste sucht sie jetzt wieder das Wort, das alphabetisch nach vorne gehört und setzt es auf Platz 2 der neuen Liste usw.
Da ihr die Arbeit bald zu mühsam wird, will sie den Computer für sich ordnen lassen. Die Struktur des Programms ist ihr klar:

```
' Olgas Ordnen
DIM SHARED wort$(100)
DIM SHARED n
eingabe
ordnen
ausgabe
```

Die Prozedur **eingabe** ist nahe liegend:

```
SUB eingabe
  INPUT "Wie viele Wörter?", n
  FOR i=1 TO n
    INPUT wort$(i)
  NEXT i
END SUB
```

Die eigentliche Schwierigkeit liegt in der Prozedur **ordnen**. Zunächst muss man wissen, dass die Zeichen < und > auch zwischen Wörter gesetzt werden können, und dann die alphabetische Reihenfolge betreffen.
Olga will zunächst den Computer so arbeiten lassen, wie sie es selbst gemacht hat. Ein Trick der Informatiker nimmt ihr aber viel Mühe ab: Statt mit zwei Feldern zu arbeiten, nämlich dem ungeordneten, das immer kürzer wird und dem geordneten, das immer länger wird, ist es wesentlich geschickter, mit nur einem Feld zu arbeiten, dessen Länge gleich bleibt. Wenn das Wort "array" jetzt an die Stelle 1 gesetzt wird, so muss das Wort "Wertzuweisung" dort den Platz räumen. Die einzige Möglichkeit dazu ist, dass wir "Wertzuweisung" auf den ehemaligen Platz von "array" setzen. So haben wir weiterhin alle Wörter im Feld, aber "array" steht auf dem richtigen Platz. Die ungeordnete Restliste steht auf den Plätzen 2 bis n. Zu dieser Restliste suchen wir wieder das alphabetisch erste Wort und vertauschen es mit dem Wort auf Platz 2. Die Restliste geht dann von Platz 3 bis Platz n.

Olga beginnt zu tippen

```
SUB ordnen
  FOR i=1 TO n
    setzeRichtigesWortauf(i)
  NEXT i
END SUB
```

Das heißt: Das Feld wird geordnet, indem man nacheinander für jedes i zwischen 1 und n das Wort sucht, das auf Platz i gehört, und es auf diesen Platz setzt. Wie das geht, haben wir schon angedeutet: Wir suchen in der Restliste (Platz i bis n) das alphabetisch vorderste Wort und setzen es auf Platz i. Das alte Wort auf Platz i muss den frei werdenden Platz einnehmen.

```
SUB setzeRichtigesWortauf(i)
  min=i
  FOR k=i TO n
    IF wort$(k)<wort$(min) THEN min=k
  NEXT k
  zwischenlager$=wort$(i)
  wort$(i)=wort$(min)
  wort$(min)=zwischenlager$
END SUB
```

Mit **min** ist die Stelle bezeichnet, wo das alphabetisch vorderste Wort steht. Diese Stelle wird berechnet, anschließend werden die Wörter der Plätze **i** und **min** vertauscht. Statt der hier gewählten Methode kann man in Q-BASIC auch den Befehl SWAP verwenden.

12) Schreibe die fehlende Prozedur **ausgabe**, verbessere die Bildsschirmgestaltung und ordne dann Olgas Liste.

13) Das Programm hat den Nachteil, dass man zu Beginn die Anzahl der Wörter eingeben muss. Schreibe ein Programm, bei dem die Wörter eingelesen werden, bis nur ein RETURN getippt wird (mit **UNTIL**-Schleife). Du musst dazu das Feld recht hoch dimensionieren, damit die Wörter auf jeden Fall Platz haben, und während der Eingabe die Wörter zählen. Die Anzahl wird in der Variablen **n** gespeichert, dann kann das bisherige Programm übernommen werden.

14) Schreibe ein Menü-Programm, das es erlaubt, immer wieder neue Wörter in die Liste aufzunehmen, dann wieder neu zu ordnen, auszugeben usw.

```
        M E N Ü

Eingabe von Wörtern....e
Ordnen der Liste.......o
Ausgabe der Liste......a
Drucken der Liste......d
Ende...................x

    Deine Wahl ?
```

15) Ergänze durch einen Menü-Punkt

```
Tilgen.................t,
```

der es erlaubt, ein Stichwort zu tilgen.

6.2 Dateien (Files)

Ablegen eines Feldes in eine Datei

Olga hat ihr Stichwortverzeichnis geordnet ausgedruckt. Sobald sie den Computer ausschaltet, ist die Liste nur noch auf Papier gespeichert. Will sie die Liste erweitern, so muss die ganze Liste erneut eingetippt werden.
Sicher kennst du schon die Möglichkeit, Informationen auf Diskette in einer Datei (einem **FILE)** zu speichern. Dieses File braucht einen Namen und eine Nummer. Als Namen wählen wir **stiworte**, als Nummer z.B. die Zahl 1. Dann lautet die Prozedur

```
SUB speichern
  OPEN "stiworte.dat" FOR OUTPUT AS #1
  PRINT #1, n
  FOR i=1 TO n
    PRINT #1, wort$(i)
  NEXT i
  CLOSE #1
END SUB
```

In Zeile 2 wird die Datei geöffnet. In Zeile 3 legen wir als Erstes die Anzahl der Wörter in der Datei ab. Dann werden die Wörter des Feldes nacheinander abgelegt (es ist also nicht nötig, dass die Elemente eines Files alle von derselben Art sind, wir haben zuerst eine Zahl, dann Strings abgespeichert). Am Schluss wird die Datei wieder geschlossen. Manche Leute werden durch das Schlüsselwort **OUTPUT** verwirrt, da wir doch Daten in die Datei eingeben. Das Wort ´OUTPUT´ bekommt dadurch einen Sinn, dass das Programm die Daten aus dem Computer an die Datei ausgibt.

Auslesen einer Datei in ein Feld

Das obige Vorgehen ist natürlich nur sinnvoll, wenn wir die auf Diskette gespeicherten Wörter wieder in den Arbeitsspeicher bringen können. Die entsprechende Prozedur lautet:

```
SUB laden
  OPEN "stiworte.dat" FOR INPUT AS #2
  INPUT #2, n
  FOR i=1 TO n
    INPUT #2, wort$(i)
  NEXT i
  CLOSE #2
END SUB
```

In Zeile 2 wird das File **stiworte** geöffnet, und zwar mit der Absicht, aus diesem File Daten zu holen. (Schlüsselwort **INPUT**). Die Filenummer 2 ist hierbei von uns willkürlich gewählt.
Zunächst holt man die Zahl n (Schlüsselwort **INPUT**), die ganz vorne in File steht und angibt, wie viele Wörter gespeichert sind. Dann holt man diese Wörter und speichert sie in das Feld **wort$**. Anschließend wird die Datei 2 geschlossen. Sind die Wörter im Feld **wort$** gespeichert, so können sie wie bisher behandelt werden.

16) Ergänze Olgas Programm um die beiden obigen Prozeduren, sodass sich folgendes Menü ergibt:

```
      M E N Ü


  Laden der Liste........l
  Eingabe von Wörtern....e
  Tilgen eines Wortes....t
  Ordnen der Liste.......o
  Ausgabe der Liste......a
  Drucken der Liste......d
  Speichern der Liste....s
  Ende...................x
```

Nach dem Starten des Programms muss der Benutzer die Liste laden oder (falls das Programm zum ersten Mal benutzt wird) Worte eingeben. Die Prozedur **eingabe** muss dabei so aussehen, wie in Aufgabe 13 beschrieben. Bei der Dimensionierung von dem Feld **wort$** muss eine recht große Obergrenze angegeben werden, da man nicht weiß, wie viele Wörter eingegeben werden. Ein Vorschlag, wie ein einfaches Programm aussehen könnte, ist auf Diskette abgelegt.

17) Olgas Freund Theo interessiert sich wenig für Stichworte der Informatik. Er sieht aber sofort, dass sich Olgas Programm mit wenigen Abänderungen in eines verwandeln lässt, das die Telefonnummern seiner Freunde verwaltet. Das Menü ist dasselbe, anstelle von Worten verwaltet er Datensätze, die aus Name, Vorname, Telefonnummer bestehen. Diese legt er in drei Feldern **name**$, **vorname**$, **nummer**$ ab. Die Liste wird folgendermaßen auf Diskette gespeichert:

```
SUB speichern
  OPEN "telefon.dat" FOR OUTPUT AS #1
  PRINT #1, n
  FOR i=1 TO n
    PRINT #1, nachname$(i)
    PRINT #1, vorname$(i)
    PRINT #1, nummer$(i)
  NEXT i
  CLOSE #1
END SUB
```

Schreibe Olgas Programm für Theos Zwecke um.

7 Rekursion

7.1 Rekursive Funktionen

Die Klasse 9 will ein Tischtennisturnier veranstalten. Es haben sich 6 Personen gemeldet und es soll "jede gegen jede" spielen. Wie viele Spiele sind das insgesamt? Anja und Robert versuchen das gerade zu berechnen, als noch eine Nachmeldung von Udo eintrudelt. Jetzt spielen also 7 Personen. Robert murrt: "Jetzt müssen wir neu überlegen." Anja widerspricht: "Nein, wenn wir wüssten, wie viele Spiele es bei 6 Personen sind, könnten wir die Anzahl für 7 Personen leicht ausrechnen. Es kommen noch 6 Spiele dazu, nämlich die Spiele, die Udo gegen die anderen 6 zu spielen hat." Dieser Gedankengang ist überaus fruchtbar; das Problem der 7 Personen wird auf ein gleichartiges mit 6 Personen zurückgeführt. Es ist damit zwar nicht gelöst, aber vereinfacht. Man kann nun versuchen, das vereinfachte Problem auf ein noch einfacheres zurückzuführen und das zu wiederholen, bis man bei einem leicht zu lösenden Fall landet. Diese Vorgehensweise bezeichnet man als *Rekursion*.

An unserem einfachen Beispiel wollen wir die rekursive Lösung durchführen. Dazu vereinbaren wir die Kurzschreibweise **spiele(n)** für die Anzahl der Spiele, die sich bei n Personen ergibt. Wir suchen also **spiele(7)**. Die Entdeckung von Anja lautet in unserer Schreibweise kurz:

spiele(7) = 6 + spiele(6).

Genauso kannst du dir überlegen: spiele(6) = 5 + spiele(5). Die Begründung ist dieselbe wie oben: Haben 5 Personen schon alle Spiele gegeneinander gespielt (die Anzahl ist spiele(5)), und kommt die sechste Person hinzu, so müssen noch 5 Spiele gespielt werden.

Wie du siehst, kann man den Gedankengang für eine beliebige Anzahl n verallgemeinern. Wir schreiben

spiele(n) = n - 1 + spiele(n-1) für n>2.

Eine solche Formel heißt Rekursionsformel. Wir wollen versuchen, damit unsere gesuchte Zahl **spiele(7)** auszurechnen.

spiele(7) = 6 + spiele(6)
spiele(6) = 5 + spiele(5)
spiele(5) = 4 + spiele(4)
spiele(4) = 3 + spiele(3)
spiele(3) = 2 + spiele(2)

Die Frage nach **spiele(7)** wird durch die Formel nicht beantwortet, sondern zurückgeführt auf die Frage nach **spiele(6)**, usw. Trotzdem ist die Formel nicht nutzlos, nach genügend vielen Anwendungen der Formel landen wir bei **spiele(2)**, also bei der Frage, wie viel Spiele bei zwei Personen notwendig sind. Das ist so klar, dass wir es bis jetzt gar nicht aufgeschrieben haben; bei zwei Personen ist *ein* Spiel nötig:

spiele(2) = 1.

Damit können wir in die letzte Gleichung gehen und erhalten

spiele(3) = 2 + spiele(2) = 2 + 1.

So berechnen wir nacheinander **spiele(4)**, **spiele(5)**, usw. und erhalten schließlich

spiele(7) = 6 + 5 + 4 + 3 + 2 + 1 = 21

1) Berechne die Anzahl von Spielen, die bei 10 Personen nötig sind, also die Zahl **spiele(10)**.

Für die Berechnungen genügte uns die Rekursionsvorschrift

spiele(2) = 1
spiele(n) = n - 1 + spiele(n - 1) für n>2.

In Q-BASIC können wir diese Vorschrift in der Form einer Funktionsdefinition eingeben

```
FUNCTION spiele(n)
 IF n = 2 THEN
  spiele = 1
 ELSE
  spiele = n-1 + spiele(n-1)
 ENDIF
END FUNCTION
```

Die Funktion **spiele** ist hierbei *rekursiv* definiert: **spiele(n)** ist mit Hilfe von **spiele(n - 1)** erklärt. Q-BASIC berechnet z.B. **spiele(7)** ganz ähnlich wie wir es eben gemacht haben. Wir haben uns dabei einige Gleichungen merken müssen, bis wir zu unserem Ziel kamen. Auch Q-BASIC muss sich einiges merken, und legt diese Dinge in einem besonderen Speicherbereich ab, dem *Stapel* (engl. stack). Falls die Aufgabe zu viele Rekursionsschritte benötigt und der Stapel nicht mehr ausreicht, kommt es zur Fehlermeldung "Stapelüberlauf" (stack overflow).

2) Gib die obige Funktionsdefinition in Q-BASIC ein und teste die Funktion im Direktmodus:

```
PRINT spiele(7)
PRINT spiele(100)
PRINT spiele(1000000)
PRINT spiele(0)
```

Schreibe ein Programm, das eine Tabelle der Funktionswerte aufstellt. Vielleicht findest du eine Gesetzmäßigkeit in der Tabelle (und damit eine einfachere Möglichkeit, die Funktionswerte zu berechnen).

3) Beim Aufruf im Direktmodus

```
PRINT spiele(0)
```

erwartet Robert das Ergebnis 0, da ja bei 0 Teilnehmern auch 0 Spiele stattfinden. Was antwortet der Computer? Versuche diese Antwort zu verstehen, indem du die Rekursionsvorschrift nachvollziehst.. Ändere die Definition der Funktion ab, sodass auch **spiele(1)** und **spiele(0)** etwas ´Vernünftiges´ ergeben.

Falls viele Personen an einem Turnier teilnehmen wollen, spielt man oft nach dem ´K.o.-System´: Nur wer ein Spiel gewinnt, darf weiterspielen. Wir formulieren dieses Verfahren rekursiv: Um den Gewinner einer Gruppe zu ermitteln, teilt man die Gruppe in zwei Hälften, ermittelt in jeder der Hälften den jeweiligen Sieger, und lässt diese beiden gegeneinander spielen. Wer gewinnt, ist der Gesamtsieger der Gruppe. Für die Hälften der Gruppe gilt wieder dieselbe Vorschrift, wir halbieren also die Hälften, usw. Dies können wir so oft durchführen, bis wir bei 2 Personen landen, dies ist wieder unser Ausstieg aus der Rekursion. Bezeichnen wir die Anzahl der benötigten Spiele mit **koSpiele(n)**, so gilt die Vorschrift:

koSpiele(2) = 1
koSpiele(n) = 2*koSpiele(n/2) + 1 für n = 4, 8, 16, . . .

In Worten: die Anzahl der Spiele ergibt sich aus der Anzahl der Spiele in den beiden Hälften plus 1 (für das Entscheidungsspiel). Dieses Verfahren klappt natürlich nur für die Zahlen 2, 4, 8, 16, ..., bei denen alle entstehenden Gruppen wieder halbiert werden können.

4) Bestimme mit Bleistift und Papier **koSpiele(16)**.

5) Gib die entsprechende Funktionsdefinition in Q-BASIC ein:

```
FUNCTION koSpiele(n)
  IF n = 2 THEN
   koSpiele = 1
  ELSE
   koSpiele = 1 + 2*koSpiele(n/2)
  ENDIF
END FUNCTION
```

Lasse dir im Direktmodus einige Funktionswerte ausgeben. Gib versuchsweise auch ein:

```
PRINT koSpiele(15)
```

Kannst du das Verhalten des Rechners deuten?

Erstelle ein Programm, das die Funktionswerte koSpiele(2), koSpiele(4), koSpiele(8), . . . ausgibt. Kannst du eine Gesetzmäßigkeit erkennen?

Die Funktion **koSpiele** aus Aufgabe 5 hat noch den Mangel, dass sie nur für die Zweierpotenzen 2, 4, 8, 16,... vernünftige Werte liefert. Wir haben uns bis jetzt auch nicht überlegt, was wir machen wollen, wenn die Anzahl **n** einer Gruppe nicht gerade ist. In diesem Fall ist **n/2** keine ganze Zahl. Das ist von der Bedeutung her sinnlos und ergibt beim Programmablauf einen Fehler. Wir zerlegen deshalb eine ungerade Zahl in zwei ganze Zahlen, die möglichst dicht bei der Hälfte liegen. Dies gelingt mit der ganzzahligen Division (in Q-BASIC mit Schlüsselwort **INT**). Wir zerlegen die Zahl n in die beiden Zahlen

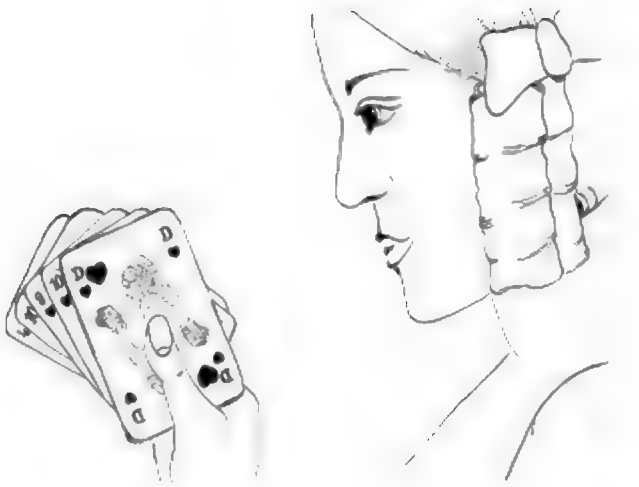
INT(n/2) und n - INT(n/2)

Falls n gerade ist, erhalten wir natürlich wie bisher die Einteilung in zwei gleich große Gruppen. Wenn wir uns diese Aufteilung immer weiter durchgeführt denken, enden wir schließlich bei Gruppen von zwei oder drei Personen. Bei zwei Personen genügt wie bisher ein Spiel, um den Gewinner festzustellen; bei drei Personen wollen wir drei Spiele abhalten. Damit lautet unsere verbesserte Funktion

```
FUNCTION koSpiele(n)
 IF n = 2 THEN
  koSpiele = 1
 ELSEIF n = 3 THEN
  koSpiele = 3
 ELSE
  koSpiele = 1 + koSpiele(INT(n/2))+ koSpiele(n-INT(n/2))
 END IF
END FUNCTION
```

6) Überlege dir zunächst ohne Computer, wie nach dieser Definition die Funktionswerte von 2, 3, 4, 5, 6,7,. . . lauten. Lasse dann den Computer rechnen und vergleiche. Schreibe ein Programm, das eine Tabelle der Funktionswerte ausgibt. Vergleiche die Tabellenwerte mit denen aus Aufgabe 5. Kannst du eine Gesetzmäßigkeit erkennen?

Beim nebenstehenden Bild hat eine Dame fünf Karten in der Hand. Auf wie viele Arten kann sie diese Karten anordnen?



Auch diese Frage können wir rekursiv anpacken. Wie stellen uns vor, wir wüssten schon, wie viele Anordnungen bei vier Karten möglich sind und nennen diese Anzahl **anordnungen(4)**. Wenn eine Anordnung von vier Karten vorliegt, gibt es fünf Möglichkeiten, eine fünfte Karte einzufügen. Insgesamt gibt es also fünfmal so viele Möglichkeiten fünf Karten anzuordnen. Wir schreiben das als:

anordnungen(5) = 5*anordnungen(4)

Entsprechende Überlegungen können wir für **n** Karten machen. Als Ausstieg aus der Rekursion nehmen wir den einfachsten Fall: Eine Karte kann nur auf eine Art angeordnet werden. In formaler Schreibweise erhalten wir als Berechnungsvorschrift:

anordnungen(1) = 1
anordnungen(n) = n*anordnungen(n-1) für n>1.

7) Berechne anhand dieser Vorschrift einige Funktionswerte. Definiere dann eine entsprechende Funktion **anordnungen** in Q-BASIC und lasse dir eine Tabelle ausgeben. Vergleiche mit der Fakultätsfunktion.

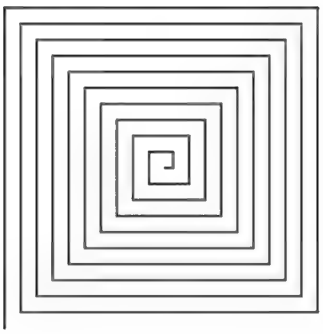
7.2 Rekursive Prozeduren

Die unten stehende Spirale kann mit dem Igel mit Hilfe einer FOR- oder WHILE-Schleife gezeichnet werden. Eine weitere Möglichkeit ist, das Zeichnen der Spirale *rekursiv* zu beschreiben. Der Grundgedanke lautet: Soll eine Spirale gezeichnet werden, so zeichnet man zunächst eine Strecke, dreht dann den Igel um 90 Grad und zeichnet anschließend eine Spirale kleinerer Größe. Ein Q-BASIC-Programm, das diesen Grundgedanken verwirklicht, ist hier angegeben.

```
' Spirale
igel

spirale 80

SUB spirale(a)
  vw a
  re 90
  IF a>2 THEN spirale a-2
END SUB
```

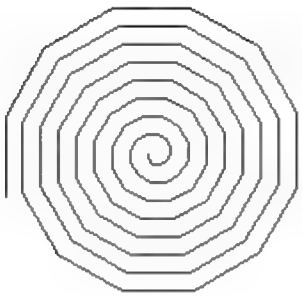
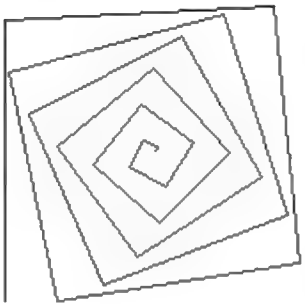
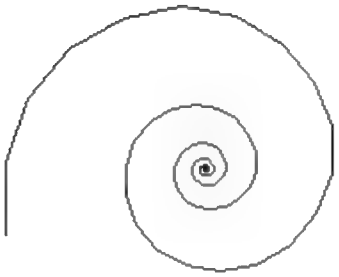


Die Größe der Spirale wird durch den Parameter **a** bestimmt. Die Prozedur ruft sich selbst auf, allerdings mit einem kleineren Parameter. Damit das Zeichnen irgendwann endet, ist der Rekursionsausstieg eingeplant.

8) Gib das obige Programm ein. Zeichne Spiralen verschiedener Größe.
9) Was geschieht, wenn der rekursive Prozeduraufruf am Anfang der Prozedur steht?

```
SUB spirale(a)
  IF a>2 THEN spirale a-2
  vw a
  re 90
END SUB
```

10) Ändere das Programm aus Aufgabe 8 ab, um verschiedene Formen von Spiralen zu bekommen. Man kann z.B. den Parameter auch durch Multiplikation mit einer festen Zahl verkleinern bzw. vergrößern.

11) Das folgende Programm verwendet die Koordinatengrafik und den Q-BASIC-Befehl **CIRCLE**. Dabei bewirkt **CIRCLE (x,y), r** das Zeichnen eines Kreises um den Punkt (x,y) mit Radius r.

```
´ Kreisfigur
SCREEN 12
WINDOW (-320,-240)-(319,239)
kreisfigur 0,0,200

SUB kreisfigur(x,y,r)
  CIRCLE(x,y),r
  IF r>2 THEN kreisfigur x+r,y,r/2
END SUB
```

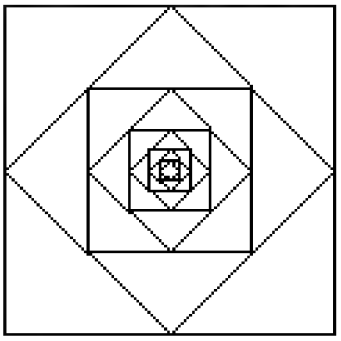
Versuche anhand des Programmes vorauszusagen, welche Zeichnung erstellt wird.

12) Schreibe ein Programm, das dieselbe Zeichnung ohne Rekursion erzeugt. Verwende eine WHILE-Schleife.

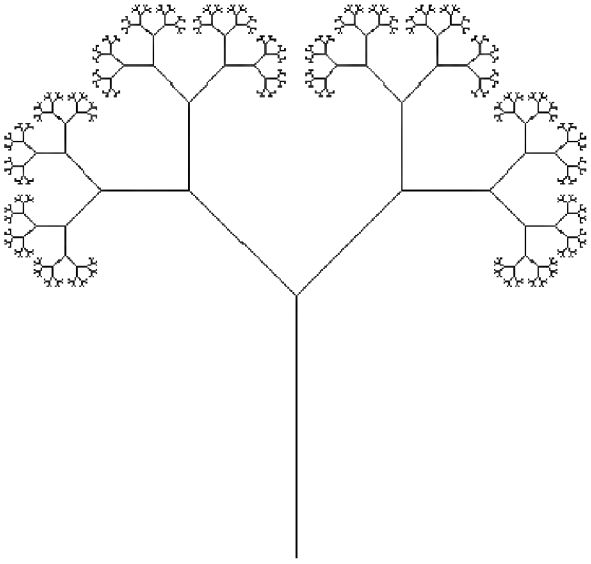
13) Ersetze die Kreise in Aufgabe 11 durch Quadrate oder Rauten.

14) Eine Zielscheibe kann rekursiv beschrieben werden als ein Kreis, der eine kleinere Zielscheibe enthält. Schreibe eine entsprechende rekursive Prozedur.

15) Schreibe ein Programm, das die nebenstehende Zeichnung erzeugt.



Der nebenstehende Baum hat die Eigenschaft, dass an jeder Gabelung zwei kleinere Bäume ansetzen. Dies deutet darauf hin, dass hier eine rekursive Prozedur günstig ist. Wir verwenden die Igelgrafik:



```
SUB baum(a)
  vw a
  IF a>1 THEN
    li 45
    baum 0.5*a
    re 90
    baum 0.5*a
    li 45
  END IF
  rw a
END SUB
```

Das Neue im Vergleich zu den vorigen Programmen ist, dass die Prozedur **baum** sich zweimal selbst aufruft. Die Prozedur ist so aufgebaut, dass sich der Igel am Ende genau an der selben Stelle und in derselben Richtung befindet wie zu Beginn.

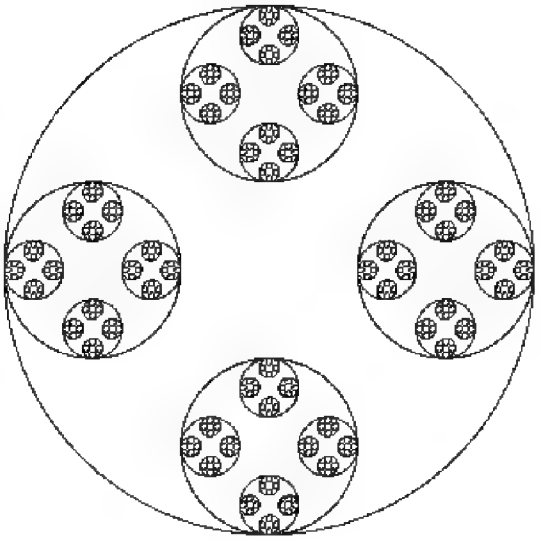
16) Versuche mit Papier und Bleistift selbst den Igel zu spielen. Überlege, was der Igel bei den folgenden Befehlen macht:

```
baum 1
baum 2
baum 4
```

17) Schreibe ein Programm, das die obige Prozedur **baum** anwendet (vergleiche Aufgabe 8). Ersetze in den Zeilen 120 und 140 den Faktor 0.5 durch einen anderen Faktor.

18) Die nebenstehende Figur besteht aus einem Kreis, der vier kleinere Kreise enthält, diese enthalten wiederum vier Kreise usw. Schreibe eine rekursive Prozedur **figur**, die diese Zeichnung bewerkstelligt.

    Hinweis: Beim Benützen der Igelgrafik genügt ein Parameter, dabei ist **figur(r)** die Figur, deren größter Kreis den Radius r hat. Verwendet man Koordinatengrafik, so braucht man drei Parameter: **figur(x,y,r)** verwendet den Mittelpunkt (x,y) und den Radius r.

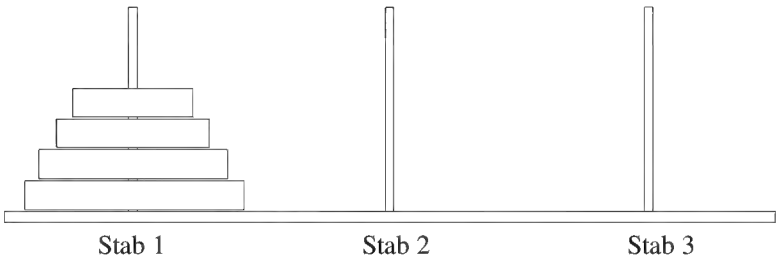


19) Lasse eine entsprechende Figur mit Quadraten zeichnen: Ein Quadrat enthält in den vier Ecken kleinere Quadrate, usw. Auch hier kann man mit Igelgrafik oder mit Koordinatengrafik arbeiten.

20) Wandle die Prozedur in Aufgabe 18 so ab, dass die vier kleinen Kreise nach außen zeigen.

Der Turm von Hanoi

Dies ist ein beliebtes Problem der Unterhaltungsmathematik, das zeigt, wie hilfreich rekursive Überlegungen sein können.



Gegeben sind drei senkrechte Stäbe. Auf dem ersten Stab sind mehrere Scheiben aufgereiht, wobei die Größe der Scheiben nach oben hin abnimmt. Es liegt also nirgends eine Scheibe auf einer kleineren Scheibe. Die Aufgabe ist, alle Scheiben auf den zweiten Stab umzuschichten. Dabei soll immer nur eine Scheibe bewegt werden, als Ablageplätze sind nur die drei Stäbe erlaubt, und nie soll eine Scheibe auf einer kleineren Scheibe liegen.

21) Versuche die Aufgabe mit vier Scheiben durchzuführen (als Scheiben kannst du vier verschieden große Münzen nehmen). Falls du das Problem nicht schaffst, versuche es zunächst mit drei Scheiben.

In der Aufgabe ist der rekursive Gedankengang zur Lösung schon angedeutet. Ich will vier Scheiben von Stab 1 auf Stab 2 umschichten. Stab 1 ist sozusagen mein Start, Stab 2 mein Ziel, Stab 3 dient mir als Zwischenlager. Nehmen wir an, ich beherrsche das Problem für drei Scheiben. Dann schichte ich die drei oberen Scheiben um, und zwar mit Stab 3 als Ziel und Stab 2 als Zwischenlager. Ist das geschehen, so liegt nur noch die größte Scheibe auf Stab 1, diese lege ich nun ins Ziel, auf Stab 2. Anschließend kann ich die drei kleineren Scheiben von Stab 3 auf Stab 2 umschichten, dabei ist nun Stab 1 das Zwischenlager.

Eine entsprechende Überlegung führt das Umschichten von drei Scheiben auf das Problem mit zwei Scheiben zurück, und dieses kann wieder auf das Problem mit einer Scheibe zurückgeführt werden. Der einfachste Fall mit einer Scheibe dient uns als Rekursionsausstieg.

```
hanoi(4,1,2,3)

SUB hanoi n,start,ziel,zwischen
  IF n>0 THEN
   hanoi n-1,start,zwischen,ziel
   legeum start,ziel
   hanoi n-1,zwischen,ziel,start
  END IF
END SUB

SUB legeum(start, ziel)
  PRINT "Lege eine Scheibe von";start, "nach", ziel
END SUB
```

22) Das Programm legt 4 Scheiben um. Ändere es ab, sodass die Anzahl der Scheiben wählbar ist.

23) Das Umlegen der Scheiben soll auf dem Grafikschirm dargestellt werden. Dazu muss die Prozedur **legeum** aufwendiger programmiert werden. Die Prozedur **hanoi** muss nicht geändert werden.

24) Es seien n Scheiben auf Stab 1. Wie oft muss man umlegen, bis man am Ziel ist? Schreibe für die benötigte Anzahl eine rekursive Funktion.

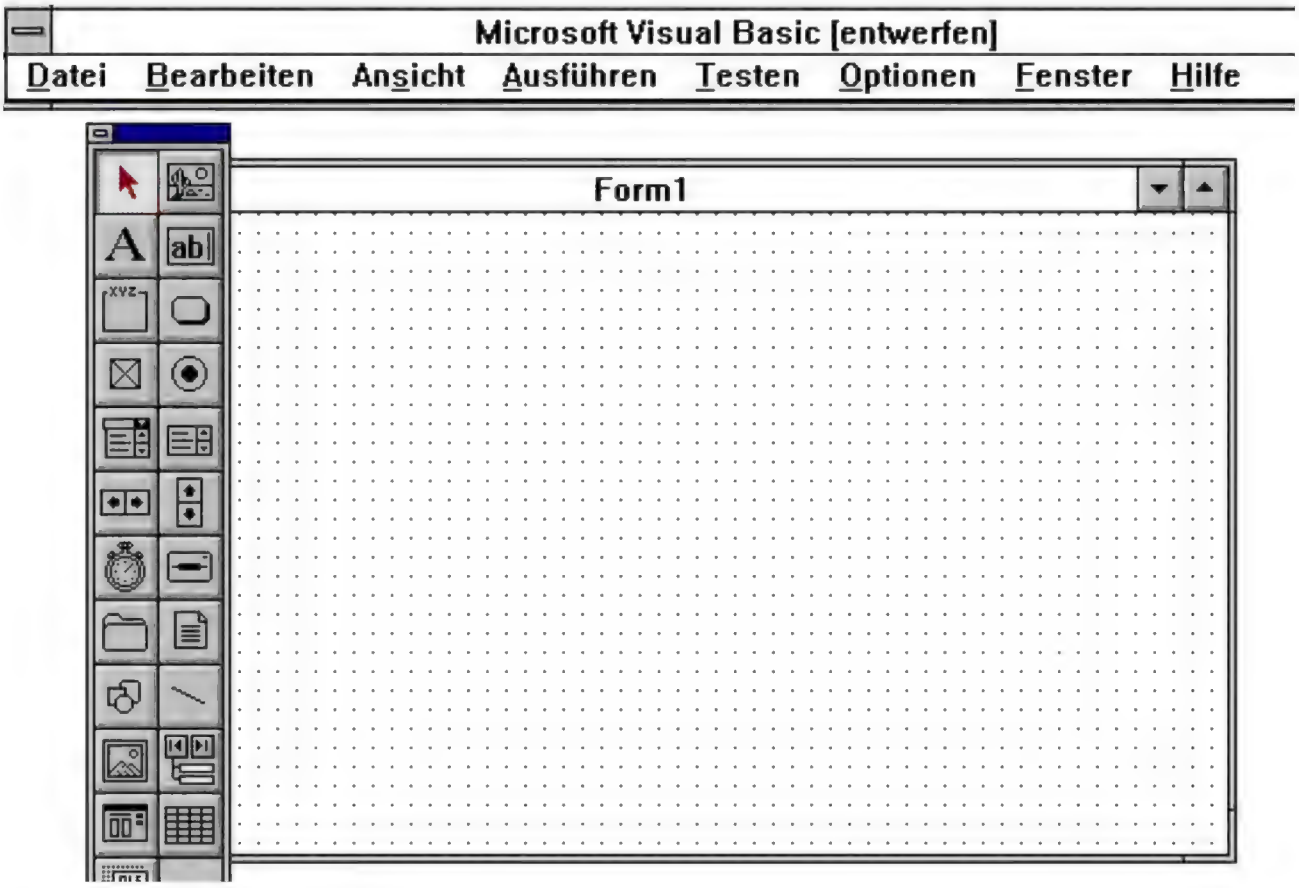
8 Ausblick auf Visual-BASIC

8.1 Die wichtigsten Objekte

All denen, die den Umstieg auf Visual-BASIC wagen wollen, soll dieses Kapitel den Übergang erleichtern. Natürlich können diese wenigen Seiten nur die allerersten Schritte in dieser neuen Sprache begleiten.

Zunächst eine gute Nachricht: Wenn du in Q-BASIC gut programmieren kannst, wird dir Visual-BASIC nicht schwer fallen; du musst nur die erste Hürde überwinden. Beide Sprachen haben nahezu denselben Wortschatz und dieselbe Grammatik, sodass du alle deine Kenntnisse weiter verwerten kannst. Es kommen allerdings einige neue Denkweisen dazu.

Wir nehmen an, dass du Visual-BASIC schon installiert hast. Nach dem Anklicken des entsprechenden Symbols kommst du nach Visual-BASIC und siehst den folgenden Bildschirm vor dir:

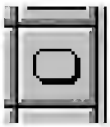


Ganz oben steht das ***Menü***, das du in ähnlicher Form schon aus Q-BASIC kennst.

Den meisten Platz auf dem Bildschirm nimmt eine so genannte ***Form*** ein, sie hat von Visual-BASIC den Namen ´Form1´ erhalten, dies könnten wir ändern, aber zunächst ist uns dieser Name gut genug. Ein Visual-BASIC-Programm kann aus sehr vielen Formen bestehen, wir werden uns meist mit einer Form begnügen. In diesem Formfenster ordnen wir die Ein- und Ausgabeelemente so an, wie sie der Benutzer später sehen soll. Einen Vorrat von solchen Elementen siehst du links in der ***Werkzeugleiste***, die wir jetzt etwas genauer untersuchen wollen. Wir wollen uns auf die allerwichtigsten Elemente beschränken.

Befehlsfelder

Gehe jetzt mit der Maus in die Werkzeugleiste und klicke das Symbol für ein ***Befehlsfeld*** an:

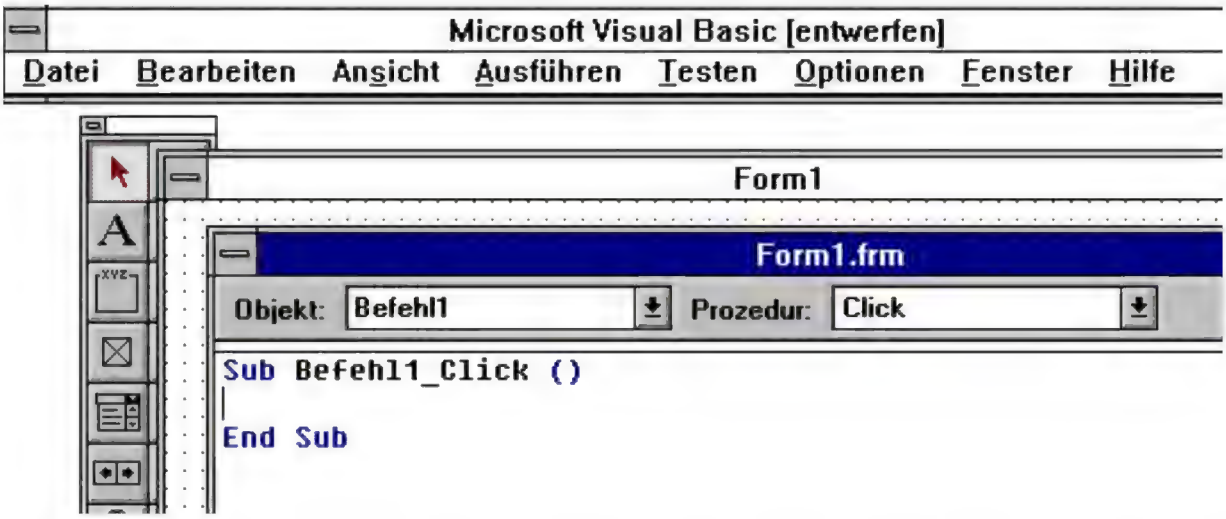


Dann gehe in die Form, drücke die linke Maustaste und ziehe bei gedrückter Taste die Maus etwas nach rechts unten.

Es entsteht eine Befehlsschaltfläche, deren Größe du bestimmen kannst.

Wenn du die Maustaste losgelassen hast, liegt die Größe und Lage fest. Du kannst beides aber jederzeit verändern. Dazu klickst du kurz das Befehlsfeld an, es ist dann markiert und zeigt am Rand acht besondere Punkte. Gehe auf einen dieser Punkte (ohne eine Maustaste zu drücken) und der Mauszeiger verändert sich und zeigt dir, in welche Richtung die Veränderung möglich ist. Wenn du jetzt bei gedrückter linker Maustaste die Maus bewegst, verändert sich die Form des Befehlsfeldes. Auch die Lage ist leicht zu verändern: Gehe erst in das Befehlsfeld hinein, und ziehe dann die Maus bei gedrückter linker Maustaste.

Nun kannst du also das Befehlsfeld an jede Stelle der Form bringen und die Größe ganz nach deinem Geschmack wählen. Visual-BASIC hat ihm auch schon eine Aufschrift gegeben, nämlich ´Befehl1´. Diese Aufschrift werden wir später noch ändern, aber zunächst wollen wir einen Befehl mit diesem Befehlsfeld verknüpfen. Dazu machst du auf dem Befehlsfeld einen Doppelklick. Das bewirkt, dass ein Fenster aufgemacht wird:



Das System hat für uns schon eine Prozedur mit dem Namen ´Befehl1_Click´ angelegt. Diese Prozedur wird ausgeführt, wenn der Benutzer das Befehlsfeld anklickt. Zurzeit steht noch nichts drin, aber du kannst dir ja einen Befehl überlegen. Wenn dir nichts Besseres einfällt, dann schreibe in die Prozedur:

```
Print "HALLO !"
```

Damit ist unser erstes Programm fertig. Starte es über den Menüpunkt ***Ausführen*** oder mit der Funktionstaste F5. Auf dem Bildschirm erscheint die Form. Sobald du das Befehlsfeld anklickst, wird der Befehl, der in der Prozedur Befehl1_Click steht, ausgeführt. Damit ist das Programm noch nicht beendet. Du kannst mehrmals das Befehlsfeld anklicken, jedes Mal wird die Prozedur Befehl1_Click ausgeführt. Das Programm können wir über den Menüpunkt ***Ausführen*** wieder beenden.

Auch Grafikbefehle können wir auf diese Weise ausführen lassen, dazu ist in Visual-BASIC keine SCREEN-Anweisung nötig, man kann sofort auf die Form zeichnen. Schreibe in die Prozedur

```
For r =100 To 2000 Step 100
 Circle (r, r), r
Next r
```

1) Erzeuge eine schöne Zeichnung auf der Form. Erkunde dazu die Grenzen deiner Form. Die Pixel sind hier wesentlich kleiner als in Q-BASIC.

Es stört noch, dass wir das Programm über das Menü beenden müssen. Wir machen deshalb ein neues Befehlsfeld auf (es hat zunächst die Aufschrift ´befehl2´), durch Doppelklick kommen wir an die Prozedur Befehl2_Click und fügen ein

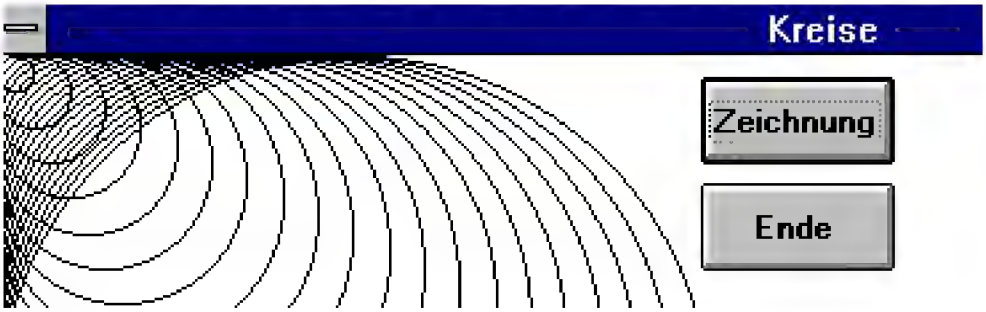
```
Sub Befehl2_Click
 End
End Sub
```

Jetzt können wir das Programm beenden, indem wir Befehl2 anklicken. Allerdings kommt spätestens jetzt der Wunsch auf, die Aufschriften auf den Befehlsfeldern zu ändern, schließlich sind diese bis jetzt für einen Benutzer nicht sehr hilfreich. Diese Aufschriften sind ***Eigenschaften*** der Befehlfelder. Um sie zu ändern gehen wir folgendermaßen vor:

- Markieren des Befehlsfeldes (Anklicken)
- Öffnen des *Eigenschaftfensters* durch Funktionstaste F4 oder über Menüpunkt Fenster
- Anklicken der Eigenschaft Caption (Aufschrift), diese ändern, mit RETURN abschließen.

2) Ändere die Aufschrift auf dem 1. Befehlsfeld ab in ´Zeichnung´, die auf dem zweiten Befehlsfeld in ´Ende´. Ändere die Überschrift (Caption) von Form1 in ´Kreise´.

Wie du bei dieser Gelegenheit siehst, hat ein Befehlsfeld sehr viele Eigenschaften, einige sprechen für sich wie Height (Höhe), Width (Breite) und eben Caption (Überschrift, Aufschrift). Eine wichtige Eigenschaft ist noch der Name. Dieser lautet weiterhin ´Befehl1´, auch wenn jetzt die Aufschrift ´Zeichnung´ auf dem Befehlsfeld steht. Auch die Namen können wir ändern, dadurch wird das Programm lesbarer. In unserem Beispiel geben wir dieselben Namen, wie wir sie als Aufschriften verwendet haben. Wenn du jetzt das Programm ablaufen lässt, ergibt sich folgendes Bild:



Speichern

Dieses Programm ist so schön, dass wir es speichern wollen. Wir müssen dir wohl nicht mehr sagen, dass du zum Menüpunkt ´Datei´ gehen musst. Dort allerdings gibt es eine ziemliche Vielfalt von Möglichkeiten. Visual-BASIC speichert nämlich das Programm (hier heißt es ***Projekt***) nicht als Ganzes ab, sondern jede Form wird einzeln abgespeichert. Falls deine Form noch den Namen ´Form1´ hat, ändere diesen jetzt ab. Gehe also nochmal zur Form, klicke diese an, hole mit F4 das Eigenschaftsfenster und ändere den Namen in ´Kreise´ um. Speichere dann das ´Projekt´ unter dem Namen ´a2´ ab. Dabei wirst du dann gefragt, ob du die Form ´Kreise´ speichern willst. Antworte mit "Ja".

Wenn du anschließend in das Verzeichnis schaust, findest du dort nicht nur die Datei ´a2.mak´ sondern auch ´kreise.frm´. Die Form ist also zusammen mit dem Projekt abgespeichert worden. Formen können auch allein abgespeichert werden mit dem Menüpunkt ´Datei speichern´.

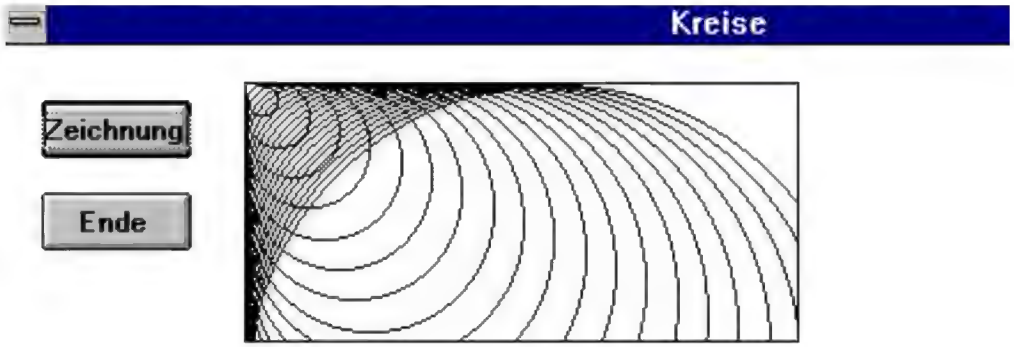
Bildfelder

Meistens zeichnet und schreibt man nicht direkt in die Form, sondern verwendet dazu Bildfelder. Wähle aus der Werkzeugleiste das entsprechende Symbol und platziere ein Bildfeld auf der Form. Mache das Bildfeld nicht zu klein.

Das Bildfeld erhält vom System den Namen ´Bild1´. Bildfelder haben keine Aufschrift (Caption). Nun sollen die Kreise nicht mehr auf die ganze Form, sondern nur noch in das Bildfeld gezeichnet werden. Das geschieht dadurch, dass man vor den Zeichenbefehlen, durch einen Punkt getrennt, den Namen des Bildfeldes angibt. In unserem Fall müssen wir die Prozedur Zeichnung_Click abändern

```
Sub Zeichnung_Click ()
 For r=100 To 2000 Step 100
  bild1.Circle (r, r), r
 Next r
End Sub
```

Wenn du das Programm startest und auf ´Zeichnung´ drückst, sollte sich folgendes Bild ergeben:



Auch dieses Programm (Visual-BASIC nennt es ´Projekt´) wollen wir speichern. Wir wollen dabei das alte Projekt ´a2.mak´ nicht überschreiben. Dann genügt es nicht, dem Projekt einen neuen Namen, etwa ´a3.mak´ zu geben. Wir müssen auch allen Formen, die sich geändert haben, neue Namen geben, da diese gesondert gespeichert werden. Bevor du das Projekt abspeicherst, benenne also die Form um, etwa in ´kreise3´, speichere diese ab mit dem Menüpunkt ´Datei speichern´ und speichere dann das Projekt mit ´Speichern unter´ ab.

3) Speichere das Projekt unter dem Namen ´a3´ ab. Verändere Form und Lage des Bildfeldes. Zeichne andere Figuren in das Bildfeld.

Das Projekt besteht nicht nur aus den Prozeduren, sondern auch aus der Form und deren Gestaltung. Du kannst jederzeit die Befehlsfelder an anderer Stelle platzieren oder das Bildfeld größer oder kleiner machen; wenn du das Prokekt abspeicherst, werden diese Änderungen bewahrt. Willst du das Projekt unter seinem alten Namen speichern, so genügt das Anklicken von ´Projekt speichern´ unter dem Menüpunkt ´Datei´. Sämtliche Formen werden dann automatisch mitgespeichert.

4) Das Bildfeld soll erst erscheinen, wenn der Benutzer auf das Befehlsfeld ´Zeichnung´ klickt. Setze dazu bei Bild1 die Eigenschaft visible auf ´False´. In die Prozedur Zeichnung_Click schreibst du dann zu Anfang

```
Bild1.Visible=TRUE
```

Textfelder und Bezeichnungsfelder

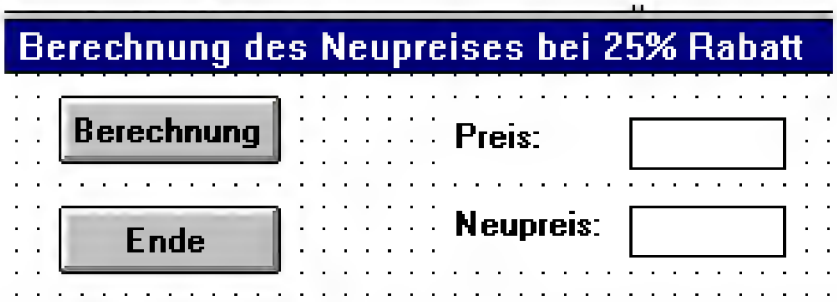
Bei unseren Q-BASIC-Programmen spielte der INPUT-Befehl für die Eingabe von Zahlen und Texten eine große Rolle. Diesen Befehl gibt es in Visual-BASIC nicht. Die Eingabe (wie auch die Ausgabe) von Zahlen und Texten erfolgt über Textfelder. Das wollen wir am Beispiel der ersten Aufgabe aus Kapitel 2 durchführen. Erinnerst du dich noch an Lehrling Ludwig. Er musste von den alten Preisen einen 25%-igen Rabatt abziehen. Wir benützen hierzu zwei Textfelder; eines für den alten Preis, das andere für den neuen.

Gehe in die Werkzeugleiste und klicke das Symbol für ein Textfeld an.
Ziehe in der Form zwei Textfelder auf. Textfelder haben keine Aufschrift (Caption). Wenn du das Eigenschaftsfenster aufmachst, siehst du, dass sie zunächst die Namen ´Text1´ bzw. ´Text2´ haben.

Zu jedem Textfeld gehört ein Text. Wie du im Eigenschaftsfenster siehst, hat das erste Textfeld den Text ´text1´ und das zweite Textfeld den Text ´text2´. Das wollen wir gleich ändern. Beide Textfelder sollen als Text zunächst gar nichts enthalten, das erreichst du durch entsprechenden Eintrag ins Eigenschaftsfenster. Als Namen geben wir den Textfeldern gemäß ihrer Bestimmung ´Preis´ bzw. ´Neupreis´.

Damit später auch der Benutzer weiß, was in den Textfeldern ist, öffne nun noch zwei Bezeichnungsfelder. Das machst du ganz entsprechend zu den anderen Feldern, das Symbol in der Werkzeugleiste ist ein ´A´.

Diese Bezeichnungsfelder tragen zunächst die Aufschrift ´Bezeichnung1´ bzw. ´Bezeichnung2´. Ändere diese Eigenschaften über das Eigenschaftenfenster in ´Preis:´ bzw. ´Neupreis:´.
Schließlich zeichnest du noch zwei Befehlsfelder in die Form. Das eine bekommt die Aufschrift ´Berechnung´, das andere ´Ende´. Gib ihnen auch die Namen ´Berechnung´ bzw. ´Ende´. .Die Oberfläche sollte dann etwa so aussehen:



Die Oberfläche ist perfekt, jetzt müssen nur noch die Prozeduren hinter der Oberfläche programmiert werden. Die Zahl, die im Textfenster ´neupreis´ steht, muss 75% des Preises betragen. wir schreiben:

```
Sub Berechnung_Click
 neupreis.text=0.75*preis.text
End Sub
```

Unsere Programmierarbeit bestand nur aus einer Zeile, diese muss allerdings verstanden werden. Wir können sie folgendermaßen lesen:
Der Text im Textfeld ´Neupreis´ ergibt sich, indem man den Text im Textfeld ´Preis´ mit 0.75 multipliziert.

5) Gestalte die Oberfläche wie angegeben, schreibe die Prozeduren Berechnung_Click und Ende_Click. Starte das Programm. Was passiert, wenn man die Berechnung anklickt und im Textfeld Preis keine Zahl steht?

8.2 Einige Projekte

Ludwigs Projekt

Es war überraschend, dass die Lösung der letzten Aufgabe mit nur einer Programmzeile gelang. Allerdings lässt das Programm doch noch einige Wünsche offen. Der größte Mangel ist, dass es abbricht, wenn die Berechnung angeklickt wird, obwohl sich noch keine Zahl im Textfeld Preis befindet. Dies wollen wir abfangen. Wir verwenden dazu die Funktion **val**, die zu einer Zeichenkette die zugehörige Zahl berechnet. Entspricht dem Anfang der Zeichenkette keine Zahl, so ergibt sich der Wert 0 (Genaueres findest du in der Hilfe). Falls sich keine Zahl im Textfeld befindet, wollen wir dem Benutzer eine Meldung geben. Dies geht in Visual-BASIC sehr einfach mit der Anweisung MsgBox. Wir ändern die Prozedur Berechnung_Click:

```
Sub Berechnung_Click ()
 z=val(preis.text)
 If z<=0 then
  MsgBox "Gültige Zahl eingeben! "
 Else
  neupreis.text=0.75*z
 End If
 Preis.SetFocus
End Sub
```

Die letzte Anweisung setzt den Cursor wieder in das Textfeld Preis. Wir wollen außerdem, dass der Neupreis verschwindet, sobald im Textfeld etwas geändert wird. Dies können wir erreichen, indem wir die Prozedur Preis_Change verwenden.

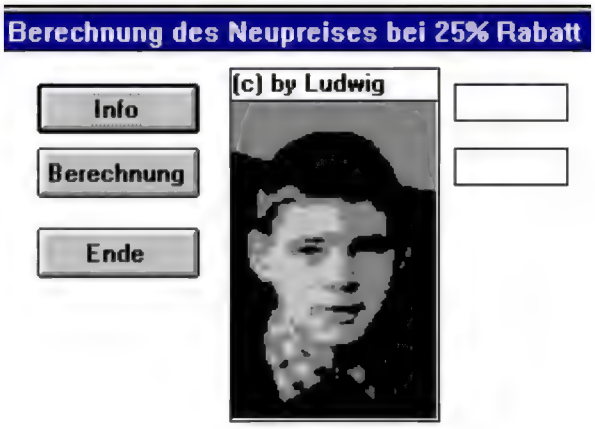
6) Ändere zunächst die Prozedur Berechnung_Click wie oben angegeben und teste ihre Wirkung, indem du unsinnige Eingaben im Textfeld Preis machst. Beende das Programm und gib dem Textfeld Preis einen Doppelklick. Es wird dir die Prozedur Preis_Change angeboten. Trage ein:

```
Sub Preis_Change ()
 Neupreis.Text =""
End Sub
```

Teste die Wirkung dieser Prozedur.

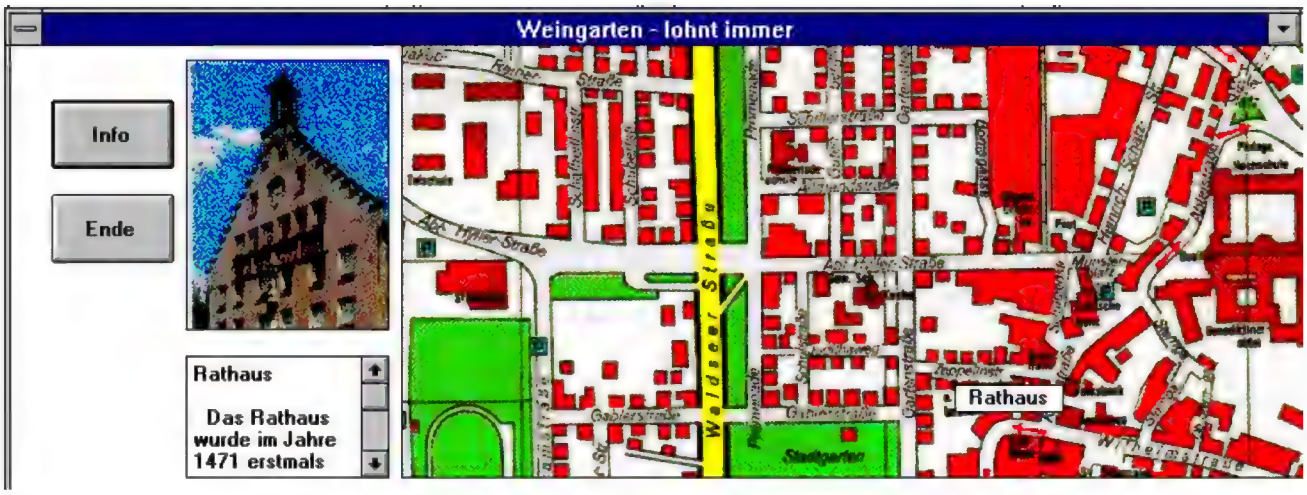
Du siehst: Außer ´Click´ werden noch andere Möglichkeiten angeboten, eine Prozedur anzustoßen. Suche selbst weitere Möglichkeiten.

7) Ludwig versieht sein Programm noch mit einem Befehlsfeld ´Info´. Wird dieses angeklickt, so erscheint ein Bildfeld mit dem Bild von Ludwig und einem Copyright-Vermerk. Wird dieses Bildfeld angeklickt, so verschwindet es wieder. Ergänze auch dein Programm entsprechend. Suche dazu in der Hilfe nach den Schlüsselwörtern ´Visible´ und ´LoadPicture´.



Der informative Stadtplan

Auf dem Bildschirm erscheint der Plan einer Stadt. Streicht man mit dem Mauszeiger darüber und kommt in die Nähe einer Sehenwürdigkeit, so erscheint deren Name in einem eingeblendeten Bezeichner. Wird dieser angeklickt, so erhält man ein Bild der Sehenwürdigkeit und in einem Textfeld weitere Informationen.



Um ein solches Projekt zu verwirklichen, entwerfe eine Form mit den abgebildeten Befehlsfeldern und einem großen Bildfeld (ich gebe ihm den Namen ´Plan´), in das ein Bild des Stadtplans übertragen wird. Dazu muss der Stadtplan als Bitmap-Datei vorliegen. Du klickst im Eigenschaftsfenster des Bildfeldes die Eigenschaft Picture doppelt an, dann kannst du den Pfad und den Namen der Bitmap-Datei auswählen.

Die wichtigste Prozedur des Projekts ist Plan_MouseMove. In dieser Prozedur kannst du die Koordinaten X und Y des Mauszeiger verwenden. Wenn du weißt, dass das Rathaus die Koordinaten X = 4500, Y= 3750 hat, schreibst du in diese Prozedur

```
Sub Plan_MouseMove(X,Y)
 If X>4800 And X< 5000 And Y>3650 And Y<3850 then
  BezAmtshaus.Visible=True
 Else
  BezAmtshaus.Visible = False
 END IF
End Sub
```

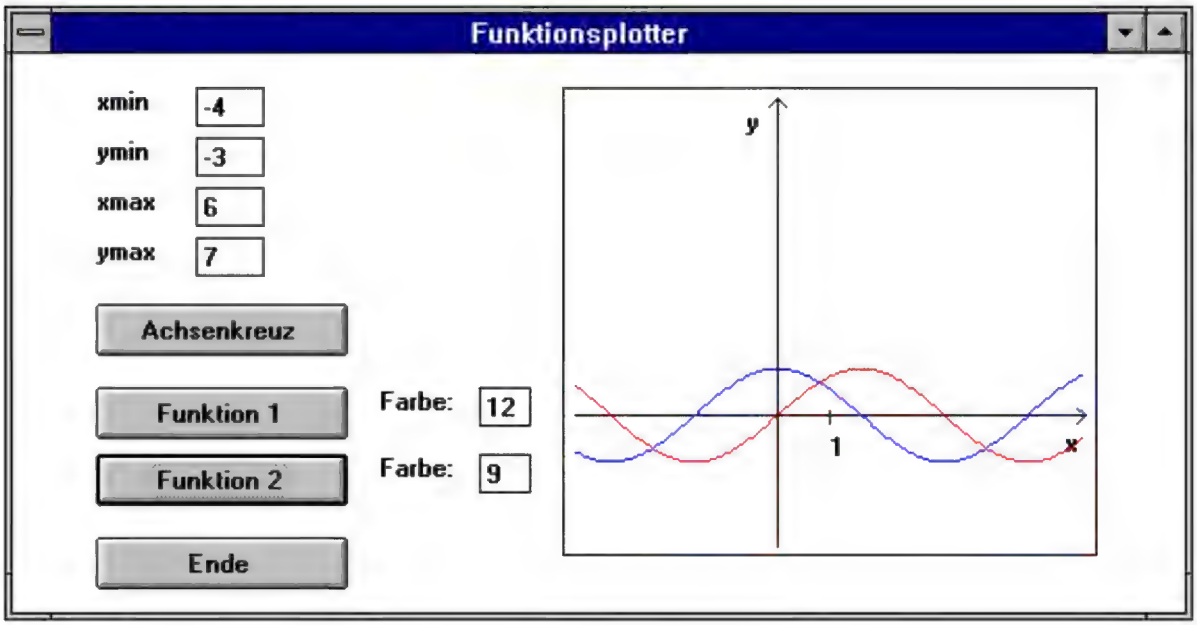
Damit das Gewünschte passiert, muss natürlich ein Bezeichner mit dem Namen BezAmtshaus existieren, dessen Eigenschaft Visible in der Entwurfsphase auf False gesetzt worden ist.

8) Schreibe das Programm für deinen Wohnort. Beim Anklicken des entsprechenden Bezeichners soll auf der linken Seite der Form im Bildfeld ein Bild der Sehenswürdigkeit und im Textfeld ein passender Text erscheinen. Der Befehl, in das Bildfeld ´Bild´ die Bitmap-Datei ´amtshaus.bmp´zu laden, lautet hierbei z.B.

```
Bild.Picture = LoadPicture("c:\amtshaus")
```

Das Bild der Sehenswürdigkeit und der zugehörige Text sollen wieder verschwinden, wenn sie angeklickt werden.

Ein Funktionsplotter



Beim Projekt eines Funktionsplotters können wir auf das entsprechende Q-BASIC-Programm aus Aufgabe 21 von Kapitel 3 zurückgreifen. Den Funktionsgraph zeichnen wir in ein Bildfeld, die erforderlichen Eingaben lassen wir mit Hilfe von Textfeldern machen.

Etwas anders läuft die Einrichtung eines internen Koordinatensystems. In Q-BASIC hatten wir hierfür den Befehl WINDOW, in Visual-BASIC schaffen wir das über die Eigenschaften ScaleLeft, ScaleTop, ScaleWidth und ScaleHeight, mit denen wir festlegen, wo das neue Koordinatensystem links bzw. oben beginnen soll und wie breit bzw. hoch es sein soll. In unserem Beispiel würden wir in die Prozedur Achsenkreuz_Click schreiben:

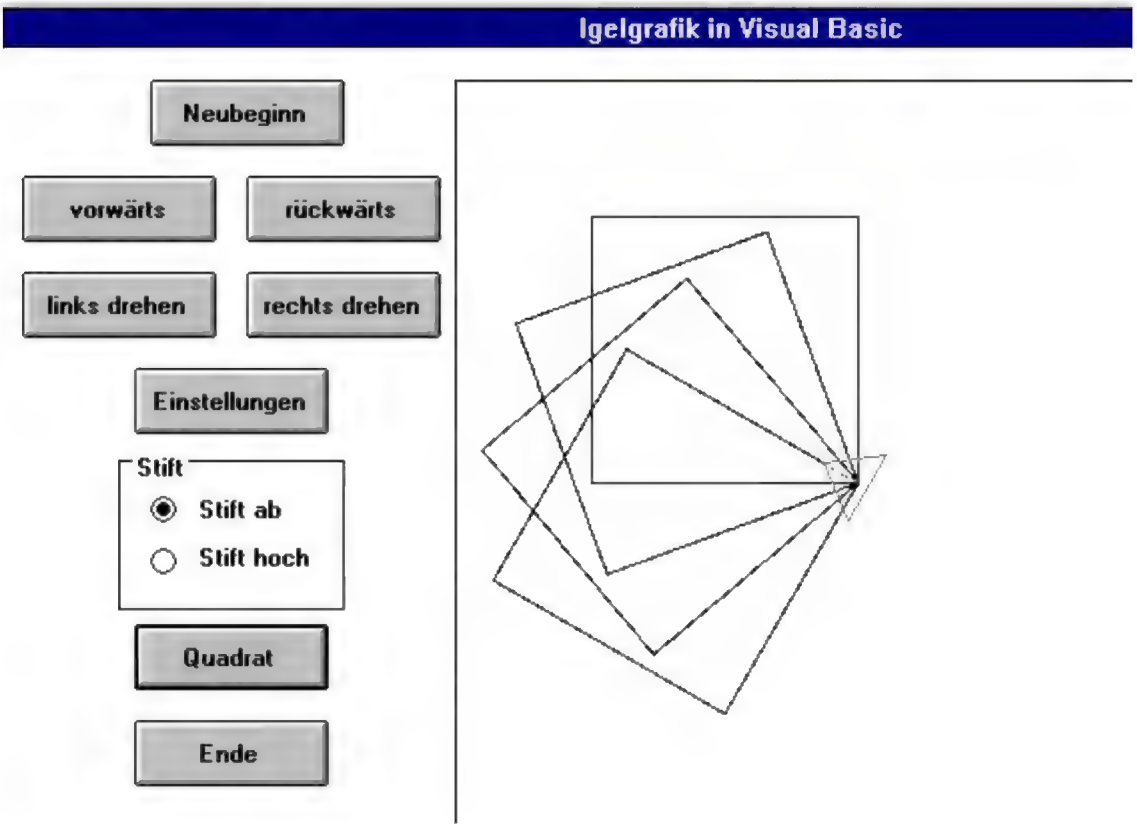
```
bild1.ScaleLeft = xmin
bild1.ScaleTop = xmax
bild1.ScaleWidth = xmax - xmin
bild1.ScaleHeight = ymin - ymax
```

Die letzte dieser Anweisungen ist nicht selbstverständlich, da hier die Eigenschaft ScaleHeight eine negative Zahl ergibt. Dadurch wird Visual-BASIC veranlasst, die y-Werte von unten nach oben zu zählen, während es sie bei positiven Werten von ScaleHeight von oben nach unten zählt.

Bei den Farben wird eine Zahl zwischen 0 und 15 erwartet. Eine solche Zahl kann man mit der Funktion QBColor umrechnen in den entsprechenden Farbwert für Visual-BASIC. Damit haben wir uns auf die 16 Farben von Q-BASIC beschränkt, was wohl für die Darstellung von Funktionsgraphen ausreicht. Falls dich die weitergehenden Farbmöglichkeiten von Visual-BASIC interessieren, kannst du einiges in der Hilfe unter dem Stichwort ´RGB´ erfahren.

9) Entwerfe das vollständige Projekt. Das Programm bricht ab, falls ein Benutzer für die Berandungswerte des Koordinatensystems unsinnige Werte eingibt. Versuche das abzufangen. Wenn unsinnige Werte oder Buchstaben eingegeben werden, soll eine entsprechende Nachricht ausgegeben werden, das Programm soll aber nicht abbrechen. Auch unsinnige Werte für die Farben sollen abgefangen werden.

Igelgrafik in Visual-BASIC



Das Igelprogramm aus Kapitel 4 können wir nach Visual-BASIC übertragen. Die einzige Schwierigkeit ist die Darstellung des Igels, die nicht analog zu Q-BASIC möglich ist. Hier können wir die Eigenschaft DrawMode verwenden. Hat diese Eigenschaft den Wert 7, so wird eine Zeichnung wieder gelöscht, wenn sie zum zweiten Mal gezeichnet wird. Wir zeichnen also den Igel (d.h. das Dreieck) in diesem DrawMode. Wenn der Igel weiterrücken soll, zeichnen wir ihn zunächst noch einmal an die alte Stelle. Dadurch wird er dort gelöscht, ohne dass in der Zeichnung Lücken entstehen. Die übrige Zeichnung wird im DrawMode 1 erstellt. Nachdem der Igel gezeichnet ist, lautet also die nächste Anweisung

```
bild1.DrawMode = 1
```

Das oben abgebildete Projekt ist für Anfänger gedacht, die noch nicht programmieren können. Sie sollen zunächst den Igel über die Befehlsfelder steuern. Beim Drücken auf ´vorwärts´ geht der Igel um 100 Schritte nach vorn, beim Drücken auf ´rechts´ dreht er sich um 10 Grad um seine Achse im Uhrzeigersinn. Die Schrittweiten können über ´Einstellungen´ geändert werden, auch die Igelgröße und die Igelgeschwindigkeit kann man ändern. Beim Drücken auf ´Einstellungen´ wird eine neue Form sichtbar, in der diese Änderungen vorgenommen werden können.

Beherrscht der Anfänger die Direktsteuerung des Igels, darf er hinter das Befehlsfeld ´Quadrat´ schauen und die Quadratprozedur abändern zu einer Dreiecks-, Fünfecksprozedur, usw. Man kann auch umfangreichere Prozeduren an diese Stelle schreiben, natürlich muss jeweils die Aufschrift des Befehlsfeldes entsprechend geändert werden. So lassen sich einfache Elemente des Programmierens vermitteln, wobei die Prozedur sofort ausprobiert werden kann und ihre Wirkung sehr anschaulich und vielleicht auch noch ästhetisch ist.

10) Entwerfe ein solches Igelprojekt.

Magic-Eye-Bilder

Visual-BASIC bietet zwei entscheidende Vorteile, wenn man Magic-Eye-Bilder erzeugen will:

- Bilder können ohne Umweg über den Bildschirm direkt zum Drucker geschickt werden
- Man kann gescannte Bilder (BMP-Dateien) verwenden,

Der erste Vorteil wird wesentlich, wenn die Tiefenwirkung sich kontinuierlich verändern soll, dann ist das Auflösungsvermögen des Bildschirms zu klein, statt eines kontinuierlichen Übergangs entsteht der Eindruck von Schichten. Schickt man die Zeichnung direkt zum Drucker (Objekt ´Printer´), so kann das bessere Auflösungsvermögen des Druckers ausgenützt werden.

Beim unten abgebildeten Projekt spielte der zweite Vorteil eine Rolle. Sowohl für das Oberflächenmuster als auch für die Tiefenfunktion haben wir eingescannte Bilder benutzt. Den handschriftlichen Text aus dem Bildfeld ´bild1´ rechts unten haben wir punktweise in die obere Form übertragen. Die Schrift ´ENDE´ aus dem linken unteren Bildfeld ´bild2´ dient zum bequemen Definieren der Tiefefunktion

```
Function tiefe(x,y)
 farbe = bild2.point(0.8*x,0.8*y)
 If farbe = 0 then
  tiefe=19
 Else
  tiefe=20
 End If
End Function
```

Anhang

Lösungen und Programmbeispiele

Lösungen zu Kapitel 2

```
'Aufgabe 2.004
CLS
INPUT "Länge in cm?  ", l
INPUT "Breite in cm? ", b
PRINT
a = l * b
PRINT "Fläche:"; a; "cm^2"
END
```

```
'Aufgabe 2.006
CLS
INPUT "Nettopreis ?  ", netto
PRINT
brutto = netto * 1.15
PRINT "Bruttopreis:"; brutto
END
```

```
'Aufgabe 2.008 Durchschnitt
CLS
INPUT "1. Note?  ", n1
INPUT "2. Note?  ", n2
INPUT "3. Note?  ", n3
PRINT
d = (n1 + n2 + n3) / 3
PRINT "Durchschnitt:"; d
END
```

```
'Aufgabe 2.011
CLS
INPUT "Zinssatz in %? ", p
INPUT "Kapital ", kapital
PRINT
zins = kapital * p / 100
endkap = kapital + zins
PRINT "Jahreszins: DM"; zins
PRINT "Endkapital: DM"; endkap
END
```

```
'Aufgabe 2.021 Rabatt
CLS
INPUT "Anzahl? ", n
FOR i = 1 TO n
 PRINT
 INPUT "Preis (in DM)?  ", preis
 rabatt = preis * .25
 neupreis = preis - rabatt
 PRINT "Neupreis: DM"; neupreis
NEXT i
```

```
'Aufgabe 2.025 Tabelle
CLS
PRINT "x", "x^2"
PRINT
FOR i = 1 TO 10
 PRINT i, i * i
NEXT i
```

```
'Aufgabe 2.028 Bildchen
CLS
FOR i = 1 TO 4
 FOR k = 1 TO i
  PRINT "*";
 NEXT k
 PRINT
NEXT i
PRINT
FOR i = 1 TO 4
 FOR k = 1 TO i
  PRINT " ";
 NEXT k
 PRINT "Hallo"
NEXT i
```

```
'Aufgabe 2.032 Gauss
CLS
PRINT "Das Programm berechnet die
Summe der Zahlen von 1 bis n"
PRINT
INPUT "n? ", n
summe = 0
FOR i = 1 TO n
 summe = summe + i
NEXT i
PRINT
PRINT "Die Summe ist"; summe
```

```
'Aufgabe 2.039 Tabelle
CLS
INPUT "Anfangskapital? ", akapital
INPUT "Zinssatz in %? ", p
INPUT "Anzahl der Jahre? ", n
PRINT
PRINT "Jahr", "Kapital"
PRINT
kapital = akapital
FOR i = 1 TO n
 zins = kapital * p / 100
 kapital = kapital + zins
 PRINT i, kapital
NEXT i
```

```
'Aufgabe 2.043 Zählen
CLS
i = 1
WHILE i <= 100
 PRINT i,
 i = i + 1
WEND
'Aufgabe 2.045 WHILE
CLS
p = 25
INPUT "Preis? ", preis
WHILE preis > 0
 rabatt = preis * p / 100
 neupreis = preis - rabatt
 PRINT "Neupreis:"; neupreis
 PRINT
 INPUT "Preis? ", preis
WEND


'Aufgabe 2.046 UNTIL
CLS
p = 25
DO
 INPUT "Preis? ", preis
 rabatt = preis * p / 100
 neupreis = preis - rabatt
 PRINT "Neupreis:"; neupreis
 PRINT
LOOP UNTIL preis = 0


'Aufgabe 2.047 Eier
CLS
i = 0
INPUT "Wie viele Eier? ", n
WHILE n >= 6
 i = i + 1
 n = n - 6
WEND
PRINT "Man braucht"; i; "Kartons"
PRINT "Es bleiben"; n; "Eier
übrig"


'Aufgabe 2.048 Gerade Zahlen
CLS
i = 2
WHILE i <= 100
 PRINT i,
 i = i + 2
WEND


'Aufgabe 2.050 Gerade Zahlen
CLS
FOR i = 2 TO 100 STEP 2
 PRINT i,
NEXT i


'Aufgabe 2.052 Tabelle
CLS
PRINT " x", "x^2"
PRINT
FOR x = -1 TO 1 STEP .25
 PRINT USING "+#.##      +
#.####"; x; x * x
NEXT x


'Aufgabe 2.054 Ungerade Z
CLS
summe = 0
INPUT "n? ", n
FOR i = 1 TO n STEP 2
 summe = summe + i
NEXT i
PRINT "Die Summe aller ungeraden
Zahlen bis"; n; "ist"; summe

'Aufgabe 2.056 Baby
CLS
INPUT "Gib dein Alter ein? ", a
IF a < 12 THEN
 PRINT "Hallo Baby"
ELSE
 PRINT "Hallo Kumpel"
END IF


'Aufgabe 2.057 Trimtrab
CLS
INPUT "Pulszahl? ", puls
IF puls > 130 THEN
 PRINT "Langsamer!"
ELSE
 PRINT "Alles  O.K."
END IF


'Aufgabe 2.058 Robert
CLS
INPUT "Temperatur? ", te
IF te < 17 THEN
 FOR i = 1 TO 20
  PRINT "Vergiss deine Jacke
nicht!"
 NEXT i
ELSE
 PRINT "Alles  O.K."
END IF


'Aufgabe 2.059 Trimtrab
CLS
INPUT "Pulszahl? ", puls
IF puls >= 130 THEN
 PRINT "Langsamer!"
ELSE
 IF puls >= 100 THEN
  PRINT "Alles  O.K."
```

```
 ELSE
  PRINT "Schneller!"
 END IF
END IF


'Aufgabe 2.063 Obergrufti
CLS
INPUT "Wie alt bist du? ", a
IF a < 12 THEN
 PRINT "Hallo Baby"
ELSEIF a < 20 THEN
 PRINT "Hallo Kumpel"
ELSEIF a < 30 THEN
 PRINT "Hallo Opa!"
ELSEIF a < 40 THEN
 PRINT "Hallo Grufti!"
ELSE
 PRINT "Hallo Obergrufti!"
END IF


'Aufgabe 2.064 Trimtrab
CLS
INPUT "Pulszahl? ", p
IF p < 50 THEN
 PRINT "Gehen Sie zum Arzt!”
ELSEIF p < 100 THEN
 PRINT "Etwas schneller bitte!"
ELSEIF p < 140 THEN
 PRINT "Gerade richtig!"
ELSEIF p < 200 THEN
 PRINT "Etwas langsamer bitte!"
ELSE
 PRINT "Geht Ihre Uhr richtig?”
END IF


'Aufgabe 2.067 Zufall
CLS
RANDOMIZE TIMER
FOR i = 1 TO 10
 PRINT RND
NEXT i


'Aufgabe 2.069 Münzenwerfen
CLS
RANDOMIZE TIMER
FOR i = 1 TO 10
 zufall = RND
 IF zufall < .5 THEN
  PRINT "Zahl"
 ELSE
  PRINT "Wappen"
 END IF
NEXT i


'Aufgabe 2.070 Münzenwerfen
CLS
RANDOMIZE TIMER
DO
 zufall = RND
 IF zufall < .5 THEN
  PRINT "Zahl"
 ELSE
  PRINT "Wappen"
 END IF
LOOP UNTIL zufall >= .5


'Aufgabe 2.073 Würfeln
CLS
RANDOMIZE TIMER
sechsen = 0
FOR i = 1 TO 60
 zufall = INT(6 * RND) + 1
 PRINT zufall,
 IF zufall = 6 THEN
  sechsen = sechsen + 1
 END IF
NEXT i
PRINT
PRINT "Anzahl der Sechsen:";
sechsen


'Aufgabe 2.074 Würfeln
CLS
RANDOMIZE TIMER
summe = 0
n = 100
FOR i = 1 TO n
 anzahl = 0
 DO
  anzahl = anzahl + 1
  zufall = INT(6 * RND) + 1
 LOOP UNTIL zufall = 6
 summe = summe + anzahl
NEXT i
PRINT
PRINT "Durchschnittliche Anzahl:";
summe / n


'Aufgabe 2.075 Würfeln
CLS
RANDOMIZE TIMER
summe = 0
n = 100
FOR i = 1 TO n
 anzahl = 0
 DO
  anzahl = anzahl + 1
  zufall = INT(6 * RND) + 1
 LOOP UNTIL zufall = 6
 IF anzahl >= 10 THEN summe =
summe + 1
NEXT i
PRINT
PRINT "Relative Häufigkeit:";summe
/ n
```

```
'Aufgabe 2.078 Verflixte Eins
CLS
RANDOMIZE TIMER
gewinn = 0
INPUT "Anzahl der Würfel? ", n
FOR i = 1 TO 100
 summe = 0
 verloren = 0
 FOR w = 1 TO n
  z = INT(RND * 6) + 1
  IF z = 1 THEN verloren = 1
  summe = summe + z
 NEXT w
 IF verloren = 1 THEN summe = 0
 gewinn = gewinn + summe
NEXT i
PRINT "Durchschnittlicher
Gewinn:"; gewinn / 100
```

```
'Aufgabe 2.079 Überbuchung
CLS
RANDOMIZE TIMER
peinlich = 0
INPUT "Wie oft wiederholen? ", n
FOR i = 1 TO n
 gekommen = 0
 FOR anmeldung = 1 TO 42
  entscheidung = INT(RND * 10)
  IF entscheidung > 0 THEN
   gekommen = gekommen + 1
  END IF
 NEXT anmeldung
 IF gekommen > 40 THEN
  peinlich = peinlich + 1
 END IF
NEXT i
PRINT "Peinliche Fälle:";
INT(peinlich / n * 100); "%"
```

```
'Aufgabe 2.081 Jahreszins
CLS
INPUT "Kapital (in DM)? ", kap
INPUT "Zinssatz (in%)?  ", p
kap2 = INT(kap)
zins = kap2 * p / 100
kap = kap + zins
PRINT "Jahreszins:"; zins; "DM"
PRINT "Endkapital:"; kap; "DM"
```

```
'Aufgabe 2.084 MWSt
CLS
INPUT "Nettopreis? ", netto
p = 15
steuer = abger(netto * p / 100)
brutto = netto + steuer
PRINT "Bruttopreis:"; brutto

FUNCTION abger (x)
 abger = INT(100 * x) / 100
END FUNCTION
```

```
'Aufgabe 2.086 Tabelle
CLS
PRINT "x", "f(x)"
FOR x = -10 TO 10
 PRINT USING "+##      +######";
x; f(x)
NEXT x

FUNCTION f (x)
 f = -4 * x * x + 2 * x
END FUNCTION
```

```
'Aufgabe 2.088 Schachtel
CLS
PRINT " x         v(x)"
PRINT
FOR x = 0 TO .5 STEP .02
 PRINT USING "#.##   #.######"; x;
v(x)
NEXT x

FUNCTION v (x)
 v = x * (1 - 2 * x) ^ 2
END FUNCTION
```

```
'Aufgabe 2.089 Schachtel
CLS
xm = 0
FOR x = 0 TO .5 STEP .001
 IF v(x) > v(xm) THEN xm = x
NEXT x
PRINT "größtes Volumen bei"; xm

FUNCTION v (x)
 v = x * (1 - 2 * x) ^ 2
END FUNCTION
```

```
'Aufgabe 2.091 Verdoppeln
CLS
PRINT " p   Verdoppelungszeit"
PRINT
FOR p = 1 TO 10
 PRINT USING "##   ###"; p;
verdopzeit(p)
NEXT p

FUNCTION verdopzeit (p)
 kap = 1
 zeit = 0
 WHILE kap < 2
  zeit = zeit + 1
  zins = kap * p / 100
  kap = kap + zins
 WEND
 verdopzeit = zeit
END FUNCTION
```

```
'Aufgabe 2.095
'Zahlenratespiel
RANDOMIZE TIMER
DIM SHARED versuchszahl AS INTEGER
CLS
anleitung
spiel
bewertung

SUB anleitung
 PRINT "Guten Tag,"
 PRINT "ich möchte mit dir
spielen."
 PRINT "Ich denke mir eine Zahl"
 PRINT "zwischen 1 und 100,"
 PRINT "du musst sie raten."
END SUB

SUB bewertung
 IF versuchszahl < 6 THEN
  PRINT "Glück gehabt."
 ELSEIF versuchszahl < 8 THEN
 PRINT "Gut geraten."
 ELSE
  PRINT "Du hast noch nicht die
richtige Methode."
 END IF
END SUB

SUB spiel
 z = 1 + INT(100 * RND)
 versuchszahl = 0
 DO
  INPUT "Rate! ", x
  versuchszahl = versuchszahl + 1
  IF x < z THEN
   PRINT "Deine Zahl ist zu
klein."
  ELSEIF x > z THEN
   PRINT "Deine Zahl ist zu groß."
  ELSE
   PRINT "Richtig geraten."
   PRINT "Du hast"; versuchszahl;
"Versuche gebraucht."
  END IF
 LOOP UNTIL z = x
END SUB

'Aufgabe 2.096
'Rechentraining
RANDOMIZE TIMER
CLS
DO
 PRINT "   M E N Ü"
 PRINT "Addition ......1"
 PRINT "Subtraktion....2"
 PRINT "Multiplikation.3"
 PRINT "Division.......4"
 PRINT "Ende...........5"
 PRINT
 INPUT "Wahl? ", wahl
 IF wahl = 1 THEN
  addition
 ELSEIF wahl = 2 THEN
  subtraktion
 ELSEIF wahl = 3 THEN
  multiplikation
 ELSEIF wahl = 4 THEN
  division
 END IF
LOOP UNTIL wahl = 5


SUB addition
 CLS
 x = 2 + INT(98 * RND)
 y = 2 + INT(98 * RND)
 PRINT x; " + "; y; " = ";
 INPUT z
 PRINT
 IF x + y = z THEN
  PRINT "Richtig"
 ELSE
  PRINT "Falsch"
 END IF
 PRINT
END SUB

SUB division
 CLS
 x = 2 + INT(8 * RND)
 y = 11 + INT(8 * RND)
 PRINT x * y; " / "; x; " = "; ""
 INPUT z
 PRINT
 IF y = z THEN
  PRINT "Richtig"
 ELSE
  PRINT "Falsch"
 END IF
 PRINT
END SUB

SUB multiplikation
 CLS
 x = 2 + INT(8 * RND)
 y = 11 + INT(9 * RND)
 PRINT x; " * "; y; " = "; ""
 INPUT z
 PRINT
 IF x * y = z THEN
  PRINT "Richtig"
 ELSE
  PRINT "Falsch"
 END IF
 PRINT
END SUB

SUB subtraktion
 CLS
 x = 19 + INT(80 * RND)
 y = 2 + INT(x * RND)
 PRINT x; " - "; y; " = "; ""
 INPUT z
```

```
 IF x - y = z THEN
  PRINT "Richtig"
 ELSE
  PRINT "Falsch"
 END IF
 PRINT
END SUB

´ Aufgabe 2.105  Teiler
DIM n AS INTEGER, i AS INTEGER
INPUT "n? ", n
PRINT
PRINT "Teiler von"; n
PRINT
FOR i = 1 TO n
 IF n MOD i = 0 THEN PRINT i, n \
i
NEXT i


´ Aufgabe 2.107  Teiler
CLS
PRINT "n", "Teilersumme"
PRINT
FOR n = 1 TO 20
 PRINT n, teilersu(n)
NEXT n

FUNCTION teilersu (n)
 su = 0
 FOR i = 1 TO n - 1
  IF n MOD i = 0 THEN su = su + i
 NEXT i
 teilersu = su
END FUNCTION


´ Aufgabe 2.108  Teilersumme
CLS
PRINT "Folgende Zahlen unter 100
sind gleich der Summe ihrer echten
Teiler:"
FOR n = 1 TO 100
 IF n = teilersu(n) THEN PRINT n
NEXT n
PRINT
PRINT "Es gibt nicht viele von der
Sorte."

FUNCTION teilersu (n)
 su = 0
 FOR i = 1 TO n - 1
  IF n MOD i = 0 THEN su = su + i
 NEXT i
 teilersu = su
END FUNCTION


´ Aufgabe 2.109
DIM x AS INTEGER, y AS INTEGER, t
AS INTEGER
CLS
INPUT "x? ", x
INPUT "y? ", y
PRINT
t = 1
FOR i = 1 TO x
 IF x MOD i = 0 AND y MOD i = 0
THEN
  t = i
 END IF
NEXT i
PRINT "ggT = "; t


´ Aufgabe 2.112 Primzahlen
DIM n AS INTEGER, i AS INTEGER
CLS
INPUT "n? ", n
i = 2
geteilt = 0
DO
 IF n MOD i = 0 THEN
  geteilt = 1
 ELSE
  i = i + 1
 END IF
LOOP UNTIL geteilt = 1 OR i >= n
IF geteilt = 0 THEN
 PRINT n; "ist eine Primzahl"
ELSE
 PRINT n; "ist keine Primzahl"
END IF


´ Aufgabe 2.113 Primzahlen
DIM n AS INTEGER, i AS INTEGER
CLS
INPUT "n? ", n
i = 2
geteilt = 0
WHILE geteilt = 0 AND i * i <= n
 IF n MOD i = 0 THEN
  geteilt = 1
 ELSE
  i = i + 1
 END IF
WEND
IF geteilt = 0 THEN
 PRINT n; "ist eine Primzahl"
ELSE
 PRINT n; "ist keine Primzahl"
END IF

´ Aufgabe 2.116 Primzahlen
CLS
PRINT "Primzahlen zwischen u und
g:"
INPUT "u? ", u
INPUT "g? ", g
FOR n = u TO g
 IF prim(n) = 1 THEN PRINT n;
NEXT n
```

```
FUNCTION prim (n)
 geteilt = 0
 IF n > 2 AND n MOD 2 = 0 THEN
geteilt = 1
 i = 3
 WHILE geteilt = 0 AND i * i <= n
  IF n MOD i = 0 THEN geteilt = 1
  i = i + 2
 WEND
 IF geteilt = 1 THEN
  prim = 0
 ELSE
  prim = 1
 END IF
END FUNCTION
```

```
´ Aufgabe 2.117 Primfaktoren
CLS
DIM n AS INTEGER, i AS INTEGER
INPUT "n? ", n
PRINT "Primfaktoren: ";
i = 2
WHILE i <= n
 IF n MOD i = 0 THEN
  PRINT i;
  n = n \ i
 ELSE
  i = i + 1
 END IF
WEND
```

```
´ Aufgabe 2.119 ggT
CLS
DIM n AS INTEGER, m AS INTEGER, i
AS INTEGER
INPUT "n? ", n
INPUT "m? ", m
i = 2
ggt = 1
WHILE i <= n AND i <= m
 IF n MOD i = 0 AND m MOD i = 0
THEN
  ggt = ggt * i
  n = n \ i: m = m \ i
 ELSE
  i = i + 1
 END IF
WEND
PRINT "ggT: "; ggt
```

```
´ Aufgabe 2.128 Zahlenfolgen
RANDOMIZE TIMER
CLS
PRINT "Welches ist die nächste
Zahl?"
a = -10 + INT(40 * RND)
b = -10 + INT(40 * RND)
FOR i = 1 TO 4
 PRINT a + b * i;
NEXT i
INPUT x
IF x = a + b * 5 THEN
 PRINT "Richtig"
ELSE
 PRINT "Falsch"
END IF
```

```
´ Aufgabe 2.131 Heron
CLS
INPUT "a? ", a
x = 1:
y = a
WHILE ABS(y - x) > .000001
 x = (x + y) / 2
 y = a / x
WEND
PRINT "Wurzel ist ungefähr"; x
```

```
´ Aufgabe 2.132 wurzel
CLS
INPUT "a? ", a
PRINT "Die Wurzel ist";
quadratwurzel(a)

FUNCTION quadratwurzel (a)
 x = 1
 y = a
 WHILE ABS(y - x) > .00000001#
  x = (x + y) / 2
  y = a / x
 WEND
 quadratwurzel = x
END FUNCTION
```

```
´ Aufgabe 2.135 Pi
CLS
INPUT "n? ", n
summe = 0
FOR i = 1 TO n
 summe = summe + SQR(1 - i * i /
(n * n))
NEXT i
a = summe / n
naeherung = 4 * a
PRINT "Näherung für PI:";
naeherung
```

```
´ Aufgabe 2.138 Pi durch
Regentropfen
CLS
SCREEN 12
WINDOW (-1, -.6)-(1.4, 1.2)
LINE (0, 0)-(1, 1), , B
CIRCLE (0, 0), 1, , 0, 3.14 / 2
INPUT "Wie viele Tropfen? ", n
anzahl = 0
```

```
FOR i = 1 TO n
 x = RND: y = RND
 PSET (x, y)
IF x * x + y * y <= 1 THEN
  anzahl = anzahl + 1
 END IF
NEXT i
verhaeltnis = anzahl / n
naeherung = 4 * verhaeltnis
PRINT "Näherung für PI:";
naeherung
```

Lösungen zu Kapitel 3

```
´ Aufgabe 3.08
SCREEN 12
WINDOW (0, 0)-(639, 479)
a = 200
x = 0
y = 0
WHILE a > 1
 LINE (x, y)-(x + a, y + a), , B
 x = x + a / 4
 y = y + a
 a = a / 2
WEND
SLEEP

´ Aufgabe 3.15
SCREEN 12
WINDOW (-14, -8)-(6, 7)
achsenkreuz
gitter
figur


SUB achsenkreuz
 LINE (-6, 0)-(6, 0)
 LINE -(5.8, .2)
 LINE (6, 0)-(5.8, -.2)
 LINE (0, -6)-(0, 6)
 LINE -(.2, 5.8)
 LINE (0, 6)-(-.2, 5.8)
 LINE (1, -.2)-(1, .2)
 LINE (-.2, 1)-(.2, 1)
 LOCATE 16, 80
 PRINT "x"
 LOCATE 3, 59
 PRINT "y"
END SUB

SUB figur
 LOCATE 1, 1
 INPUT "Anzahl der Ecken? ", n
 PRINT
 INPUT "Punkt 1:    x? ", x1
 INPUT "            y? ", y2
 PSET (x1, y1)
 FOR i = 2 TO n
  LOCATE 3, 1
  PRINT "                        "
  PRINT "                        "
  LOCATE 3, 1
  PRINT "Punkt"; i; ":",
  INPUT "x? ", x
  INPUT "               y? ", y
  LINE -(x, y)
 NEXT i
 LINE -(x1, y1)
END SUB

SUB gitter
 FOR x = -6 TO 6
  FOR y = -6 TO 6
   PSET (x, y)
  NEXT y
 NEXT x
END SUB
```

Magic-Eye-Bild

```
SCREEN 12
DIM SHARED ausg
DIM SHARED xmax
DIM SHARED ymax
DIM SHARED gleicheFarbe(639) AS INTEGER
xmax = 639
ymax = 479
WINDOW (0, 0)-(xmax, ymax)
zeichneMuster
FOR y = 0 TO ymax
 faerbeZeile y
NEXT y
```

```
SUB faerbeZeile (y)
 a = 4
 e = 20
 k = 40
 FOR x = 0 TO xmax: gleicheFarbe(x) = -1: NEXT x
 FOR x = 0 TO xmax
  t = tiefe(x, y)
  IF t < 0 THEN t = 0
  d = k * a * t / (t + e)
  d1 = INT(d)
  d2 = INT(2 * d - d1)
  IF d1 > 0 AND d2 > 0 THEN
   IF x - d1 >= 0 AND x + d2 <= xmax THEN
    IF gleicheFarbe(x + d2) > -1 THEN
     u = gleicheFarbe(x + d2): v = x - d1
     WHILE gleicheFarbe(u) > -1: u = gleicheFarbe(u): WEND
     WHILE gleicheFarbe(v) > -1: v = gleicheFarbe(v): WEND
     IF u > v THEN
      gleicheFarbe(u) = v
     ELSEIF v > u THEN
      gleicheFarbe(v) = u
     END IF
    END IF
    gleicheFarbe(x + d2) = x - d1
   END IF
  END IF
 NEXT x
 FOR x = 0 TO xmax
  IF gleicheFarbe(x) > -1 THEN
   farbe = POINT(gleicheFarbe(x), y)
   PSET (x, y), farbe
  END IF
 NEXT x
END SUB

FUNCTION tiefe (x, y)
 r2 = (x - 300) * (x - 300) + (y - 200) * (y - 200)
 IF r2 < 10000 THEN
  tiefe = 10 + .05 * SQR(10000 - r2)
 ELSE
  tiefe = 10
 END IF
END FUNCTION

SUB zeichneMuster
 FOR y = 0 TO ymax
  FOR x = 0 TO xmax
   farbe = INT(1.5*RND)*15
   PSET (x,y),farbe
  NEXT x
 NEXT y
END SUB
```

Als Muster wurde hier ein Zufallsmuster gewählt. Eine andere Möglichkeit wäre, unter den ´Tapetenmustern´ ein geeignetes auszuwählen. In der Prozedur zeichneMuster können noch zusätzlich zum Hintergrundmuster größere Objekte eingezeichnet werden, die den Augen bessere Anhaltspunkte geben als die Pixel des Zufallsmusters. Eine Möglichkeit wäre etwa, noch zusätzlich einige Quadrate auf dem Bildschirm zu verteilen.

Igelprogramm zu Kapitel 4

```
DIM SHARED t%(1 TO 410)
DIM SHARED igelx
DIM SHARED igely
DIM SHARED igelw
DIM SHARED stiftab

igel

SUB igel 'Aufruf der Igelgrafik
 SCREEN 11
 CLS
 stiftab = 1
 igelx = 320: igely = 240: igelw = 0
 GET (igelx - 10, igely - 10)-(igelx + 10, igely + 10), t%
 zeichneigel
END SUB

SUB li (w) 'links
 igelw = igelw + w * 3.141593 / 180
 vw 0
END SUB

SUB re (w)
 li (-w)
END SUB

SUB rw (a)  'rückwärts
 vw (-a)
END SUB

SUB sa 'stiftab
 stiftab = 1
END SUB

SUB sh 'stifthoch
 stiftab = 0
END SUB

SUB vw (a)   'vorwärts
 x = igelx: y = igely: w = igelw
 IF x > 10 AND x < 630 AND y > 10 AND y < 470 THEN PUT (x - 10, y - 10),
t%, PSET
 igelx = x - a * SIN(w)
 igely = y - a * COS(w)
 IF stiftab = 1 THEN LINE (x, y)-(igelx, igely)
 IF igelx > 10 AND igelx < 630 AND igely > 10 AND igely < 470 THEN
  GET (igelx - 10, igely - 10)-(igelx + 10, igely + 10), t%
  zeichneigel
 END IF
END SUB

SUB zeichneigel
 x = igelx: y = igely: w = igelw
 LINE (x - SIN(w), y - COS(w))-(x - 10 * SIN(w), y - 10 * COS(w))
 FOR i = 1 TO 3
  w1 = w + 2.094395
  LINE (x - 10*SIN(w), y - 10*COS(w))-(x - 10*SIN(w1), y - 10*COS(w1))
  w = w1
 NEXT i
END SUB
```

Erläuterung:

Die DIM SHARED-Anweisungen machen die genannten Variablen zu 'globalen' Variablen, auf die alle Prozeduren des Programms Zugriff haben.
Die wesentlichen Überlegungen stecken in der Prozedur **vw** (vorwärts). Die Grundidee ist einfach: Wir berechnen mit Hilfe trigonometrischer Überlegungen die neue Igelposition und zeichnen (falls gewünscht, d.h. die Variable *stiftab* den Wert 1 hat) eine Strecke von der alten zur neuen Igelposition. Der Igel an der alten Position ist zu löschen und an der neuen zu zeichnen. Zum Löschen des Igels verwenden wir die von QBASIC zur Verfügung gestellten Grafikanweisungen PUT und GET. Im Datensatz t%() ist ein kleiner Ausschnitt des Grafikschirms (ein Quadrat der Seitenlänge 20 um den Igel) gespeichert und zwar *ohne* Igel. Mittels PUT wird mit diesem gespeicherten Bild der alte Igel 'überschrieben', mittels GET wird die Umgebung der neuen Igelposition in t%() gespeichert und zwar speichern wir, *bevor* der Igel an der neuen Position gezeichnet wird. (Für die Parameter, welche die Anweisungen GET und PUT verlangen, schau in der Hilfe von QBASIC nach. Die Igelumgebung wird nur gespeichert und der Igel wird nur gezeichnet, wenn er auf dem Bildschirm zu liegen kommt.)
Alle Prozeduren, welche die Lage des Igels ändern (**re**, **li**, **rw**) benutzen die Prozedur **vw**. Beim Aufruf der Igelgrafik mit der Prozedur **igel**, wird erstmals die ('leere') Umgebung des Igels in t%() gespeichert.
Winkel werden gegen die Richtung 'nach oben' im Gegenuhrzeigersinn gemessen. Sie sind in Grad einzugeben und werden vom Programm sofort ins Bogenmaß umgerechnet.
Für die Prozedur **zeichneigel**, welche den Igel zeichnet, ist zu beachten, dass der Igel als ein gleichseitiges Dreieck dargestellt wird, dessen Mittelpunkt die Position des 'Zeichenstifts' ist. Die Länge der Höhe (Seitenhalbierende) wird (willkürlich) mit 15 angenommen (das entspricht einer Seitenlänge des gleichseitigen Dreiecks von rund 17).

Lösungen zu Kapitel 4

```
' Aufgabe 4.15
'Quadrate schachteln
FOR k=1 TO 8
 sh
 li 90: vw 10: re 90: rw 10
 sa
 quadrat 20*k
NEXT k

SUB quadrat(seite)
 FOR i=1 TO 4
  vw seite: re 90
 NEXT i
END SUB

' Aufgabe 4.16 a
FOR i=1 TO 10
 dreieck 20*i
NEXT i

' b: Flügel
FOR i=1 TO 4
 dreieck 100: re 90
NEXT i

' c: Wabe
FOR i=1 TO 3
 dreieck 90: re 60
 dreieck 50: re 60
NEXT i

SUB dreieck(seite)
 FOR i=1 TO 3
  vw seite: re 120
 NEXT i
END SUB

' Aufgabe 4.19
' Kreispirale
FOR r=10 TO 100 STEP 10
 halbkreis r
NEXT i

SUB halbkreis(r)
 s=3.14*4/30
 FOR i=1 TO 30
  vw s: re 6
 NEXT i
END SUB

' Aufgabe 4.24
' a: Kreisreihe
sh
li 90: vw 200: re 90: sa
FOR i=1 TO 20
 kreis 40: sh
 re 90: vw 20: li 90: sa
NEXT i
' b: Kreisring1
FOR i=1 TO 12
```

```
 sh
 vw 60: sa: kreis 60: sh
 rw 60: re 30
NEXT i

‘ c: Kreisring2
FOR i=1 to 30
 sh
 vw 80: sa: kreis 40: sh
 rw 80: re 12
NEXT i

SUB kreis(r)
 s=3.14*r/30: sh
 vw r: re 93: sa
 FOR i=1 TO 60
  vw s: re 6
 NEXT i
 sh
 li 93: rw r: sa
END SUB

‘ Aufgabe 4.25
‘ a: Blatt
sh
rw 100: li 90: vw 100: re 90
sa
FOR i=1 TO 4
 halbkreis 100
 re 90: vw 200: rw 200
NEXT i

‘b: Schustermesser
FOR i=1 TO 2
 sh
 li 90: vw 100: re 90: sa
 halbkreis 50: sh
 re 90: vw 100: re 90: sa
 halbkreis 100
NEXT i

c: yinyan
halbkreis 50: sh
li 90: vw 100: re 90: sa
halbkreis 100: halbkreis 100
halbkreis 50

SUB halbkreis(r)
‘zeichnet Rechtshalbkreis mit
‘ Position des Igels bei Prozedur-
’ aufruf als Startposition. Igel
‘endet am Ende des Halbkreises
 s=3.14*r/30: re 3
 FOR i=1 TO 30
  vw s: re 6
 NEXT i
 li 3
END SUB
```

Lösungen zu Kapitel 5

```
' Aufgabe 5.15
' Palindromprüfung
CLS
PRINT "Ein Text wird auf die Palindromeigenschaft überprüft"
INPUT "Text eingeben: ", text$
l = LEN(text$)
FOR i = l TO 1 STEP -1
 invers$ = invers$ + MID$(text$, i, 1)
NEXT i
IF text$ = invers$ THEN
 PRINT text$; " ist ein Palindrom."
ELSE
 PRINT text$; " ist  kein Palindrom."
END IF


' Aufgabe 5.27
' Chiffrieren durch "spiegeln"
CLS
INPUT "Gib die zu verschlüsselnde Nachricht ein: ", nachricht$
PRINT
code$ = gespiegelt$(nachricht$)
PRINT "Verschlüsselte Nachricht: "; code$
```

```
FUNCTION gespiegelt$ (text$)
 l = LEN(text$)
 a = ASC("a"): z = ASC("z"): ag = ASC("A"): zg = ASC("Z")
 FOR i = 1 TO l
   nr = ASC(MID$(text$, i, 1))
   IF nr >= ag AND nr <= zg THEN nr = nr - ag + a
   IF nr >= a AND nr <= z THEN
     n = nr - a ' Nummer des Buchstabens im Alphabet
     n = 25 - n 'Spiegeln
     nr = n + a  'Ascii-Code des gespiegelten Buchstabens
     zeichen$ = CHR$(nr)
     MID$(text$, i, 1) = zeichen$
    END IF
 NEXT i
gespiegelt$ = text$
END FUNCTION


' Aufgabe 5.30
' Groß- in Kleinbuchstaben, Nichtbuchstaben entfernen
CLS
PRINT "Der eingegebene Text wird komprimiert"
PRINT
INPUT "Text eingeben: ", text$
komp$ = wandelt$(text$)
PRINT "Komprimierter Text: "; komp$

FUNCTION wandelt$ (text$)
 l = LEN(text$): a = ASC("a"): z = ASC("z")
 ag = ASC("A"): zg = ASC("Z"): hilf$ = ""
 FOR i = 1 TO l
  ascii = ASC(MID$(text$, i, 1))
  IF ascii >= ag AND ascii <= zg THEN ascii = ascii - ag + a
  IF ascii >= a AND ascii <= z THEN hilf$ = hilf$ + CHR$(ascii)
 NEXT i
 wandelt$ = hilf$
END FUNCTION


' Aufgabe 5.45
' Zufallswort
l = zufallszahl(2, 8): a = ASC("a"): z = ASC("z"): wort$ = ""
FOR i = 1 TO l
 ascii = zufallszahl(a, z)
 wort$ = wort$ + CHR$(ascii)
NEXT i
PRINT "Zufallswort: "; wort$

FUNCTION zufallszahl (n, m)
 RANDOMIZE TIMER
 zufallszahl = INT(RND * (m - n + 1) + n)
END FUNCTION


' Aufgabe 5.45
' Vokale streichen
CLS
PRINT "Aus einem Text werden alle Vokale gestrichen"
PRINT
INPUT "Text eingeben: ", text$
PRINT
PRINT "Text ohne Vokale: "; gestrichen(text$)
```

```
FUNCTION gestrichen$ (text$)
 vokale$ = "aeiouAEIOU"
 l = LEN(text$): neutext$ = ""
 FOR i = 1 TO l
  z$ = MID$(text$, i, 1)
  IF INSTR(vokale$, z$) = 0 THEN neutext$ = neutext$ + z$
 NEXT i
 gestrichen$ = neutext$
END FUNCTION


' Aufgabe 5.46
' Wort innerhalb eines Textes in Großbuchstaben
DIM SHARED text$
CLS
PRINT "in einem Text wird eine bestimmte Zeichenfolge in Großbuchstaben
geschrieben."
INPUT "Welcher Text? ", dateiname$
einlesen (dateiname$)
INPUT "Welches Wort soll großgeschrieben werden? ", wort$
PRINT
text$ = gross$(wort$)
PRINT text$

SUB einlesen (dateiname$)
 OPEN dateiname$ FOR INPUT AS #1
 WHILE NOT EOF(1)
  LINE INPUT #1, zeile$
  text$ = text$ + CHR$(13) + zeile$
 WEND
 CLOSE #1
END SUB

FUNCTION gross$ (wort$)
 a = ASC("a"): z = ASC("z"): ag = ASC("A")
 w = LEN(wort$): neuwort$ = ""
 FOR i = 1 TO w
  ascii = ASC(MID$(wort$, i, 1))
  IF ascii >= a AND ascii <= z THEN ascii = ascii - a + ag
  neuwort$ = neuwort$ + CHR$(ascii)
 NEXT i
 DO
  s = INSTR(text$, wort$)
  IF s = 0 THEN EXIT DO
  MID$(text$, s, w) = neuwort$
 LOOP
 gross$ = text$
END FUNCTION


' Aufgabe 5.49
' Superhirn
DIM SHARED ratezahl$
DIM SHARED tip$
spielregel
erzeugen
n = 0    ' Zahl der Rateversuche
DO
 INPUT "Dein Tipp? ", tip$
 n = n + 1
 IF tip$ = "a" OR tip$ = ratezahl$ THEN EXIT DO
 auswertung
```

```
LOOP
IF tip$ = "a" THEN
 PRINT "Schade dass du aufgibst. Ich habe mir "; ratezahl$; " gedacht."
ELSE
 PRINT "Gratuliere. Geraten in "; n; " Versuchen."
END IF

SUB auswertung
 treffer = 0: voll = 0: halb = 0
 FOR i = 1 TO 4
  IF INSTR(ratezahl$, MID$(tip$, i, 1)) > 0 THEN
   treffer = treffer + 1
   IF MID$(tip$,i,1) = MID$(ratezahl$,i,1) THEN voll = voll + 1
  END IF
 NEXT i
 halb = treffer - voll
 PRINT voll; " Volltreffer und "; halb; " Halbtreffer"
END SUB

SUB erzeugen
 ziffer = zufallszahl(48, 57)  '48 bis 57 sind der ASCII-Code der Ziffern
 ratezahl$ = CHR$(ziffer)
 FOR i = 1 TO 3
  DO
   ziffer = zufallszahl(48, 57)
   ziffer$ = CHR$(ziffer)
  LOOP UNTIL INSTR(ratezahl$, ziffer$) = 0
  ratezahl$ = ratezahl$ + ziffer$
 NEXT i
END SUB

SUB spielregel
 CLS
 PRINT "                        SUPERHIRN"
 PRINT
 PRINT "Ich, der Computer, denke mir eine vierstellige Zahl aus
verschiedenen Ziffern,"
 PRINT "die du raten sollst. Wenn in deiner Eingabe eine Ziffer an der
richtigen Stelle"
 PRINT "sitzt, ist dies ein Volltreffer, wenn eine Ziffer erraten, aber
nicht an der "
 PRINT "richtigen Stelle sitzt, ist dies ein Halbtreffer."
 PRINT "Du gibst auf mit 'a.'"
 PRINT
END SUB

FUNCTION zufallszahl (n, m)
 RANDOMIZE TIMER
 zufallszahl = INT(RND * (m - n + 1) + n)
END FUNCTION
```

Lösungen zu Kapitel 6

```
'Aufgabe 6.01 Preisliste
DIM preis(20)
preis(1) = 15
preis(2) = 27
preis(3) = 6
preis(4) = 3
preis(5) = 11
preis(6) = 20
preis(7) = 9
CLS
FOR i = 1 TO 7
 PRINT USING "##   ###.## DM"; i;
preis(i)
NEXT i
```

```
'Aufgabe 6.02 Preisliste
DIM preis(20)
preis(1) = 15
preis(2) = 27
preis(3) = 6
preis(4) = 3
preis(5) = 11
preis(6) = 20
preis(7) = 9
CLS
i = 1
WHILE preis(i) > 0 AND i <= 20
 PRINT USING "##   ###.## DM"; i;
preis(i)
 i = i + 1
WEND

'Aufgabe 6.03 Preisliste
DIM preis(20)
DIM artbezeichnung$(20)
preis(1) = 15
preis(2) = 27
preis(3) = 6
preis(4) = 3
preis(5) = 11
preis(6) = 20
preis(7) = 9
artbezeichnung$(1) = "Fußball"
artbezeichnung$(2) =
"Hokeyschläger"
artbezeichnung$(3) = "Family-
Tennis"
artbezeichnung$(4) = "Comik-Heft"
artbezeichnung$(5) =
"Boxhandschuh"
artbezeichnung$(6) = "Puppenwagen"
artbezeichnung$(7) = "Teddy"
CLS
i = 1
WHILE preis(i) > 0 AND i <= 20
 PRINT USING "## \            \
###.## DM"; i; artbezeichnung$(i);
preis(i)
 i = i + 1
WEND

'Aufgabe 6.04 Würfeln
RANDOMIZE TIMER
DIM haeufigkeit(12)
FOR i = 1 TO 1000
 x = INT(6 * RND) + 1
 y = INT(6 * RND) + 1
 summe = x + y
 haeufigkeit(summe) =
haeufigkeit(summe) + 1
NEXT i
FOR s = 2 TO 12
 PRINT "Augensumme "; s; "fiel";
haeufigkeit(s); "mal."
NEXT s

'Aufgabe 6.07 geburtstage
RANDOMIZE TIMER
DIM geburtstage(365)
gewonnen = 0
FOR v = 1 TO 100
 FOR t = 1 TO 365
  geburtstage(t) = 0
 NEXT t
 FOR i = 1 TO 30
  tag = INT(365 * RND) + 1
  geburtstage(tag) =
geburtstage(tag) + 1
 NEXT i
 max = 0
 FOR t = 1 TO 365
  IF geburtstage(t) > max THEN max
= geburtstage(t)
 NEXT t
 IF max > 1 THEN gewonnen =
gewonnen + 1
NEXT v
PRINT "Bei 100 Versuchen: gewonnen
in"; gewonnen; "Fällen."
```

Lösungen zu Kapitel 7

```
'Aufgabe 7.03
CLS
INPUT "Wie viele Personen? ", n
PRINT "Es gibt"; spiele(n);
"Spiele."

FUNCTION spiele (n)
 IF n = 0 OR n = 1 THEN
  spiele = 0
 ELSE
  spiele = n - 1 + spiele(n - 1)
 END IF
END FUNCTION

'Aufgabe 7.07
CLS
PRINT "n     Anordnungen"
PRINT
FOR n = 1 TO 10
 PRINT USING "## #######"; n;
anordnungen(n)
NEXT n

FUNCTION anordnungen (n)
 IF n = 1 THEN
  anordnungen = 1
```

```
 ELSE

anordnungen = n * anordnungen(n-1)
 END IF
END FUNCTION
```

```
´Aufgabe 7.10b
igel
spirale 200

SUB spirale (a)
 vw a
 re 87
 IF a > 6 THEN spirale a - 9
END SUB
```

```
´Aufgabe 7.10c
igel
spirale 50

SUB spirale (a)
 vw a
 re 20
 IF a > 2 THEN spirale a * .95
END SUB
```

```
'Aufgabe 7.18 Kreisfigur
SCREEN 11
figur 300, 220, 200

SUB figur (x, y, r)
 IF r > 1 THEN
  CIRCLE (x, y), r
  figur x + 2 * r / 3, y, r / 3
  figur x, y + 2 * r / 3, r / 3
  figur x - 2 * r / 3, y, r / 3
  figur x, y - 2 * r / 3, r / 3
 END IF
END SUB
```

Kurzbeschreibung von Q-BASIC

Wichtige Tasten(kombinationen)

F1, UMSCHALTTASTE+F1	Aufruf der Hilfe
F2	Wechsel zu den Unterprogrammfenstern
F4	Wechsel zwischen Ausgabe- und Programmfenster
F5, HOCHSTELLTASTE+F5	Programmstart (Programm fortsetzen bzw. neustarten)
F6	Wechsel zwischen Programmfenster und Direktfenster
STRG+PAUSE	Unterbricht ein laufendes Programm (z.B. einen INPUT-Befehl, eine Endlosschleife)

Beim Edieren der Programme hilfreich:

UMSCHALTTASTE+ENTF	markierter Text wird 'ausgeschnitten'
UMSCHALTTASTE+EINFG	Einfügen (aus der Zwischenablage)
POS1, ENDE	an Anfang/Ende der aktuellen Zeile

Ausdruck des Grafikbildschirms

Wir setzen voraus, dass WINDOWS auf deinem Computer installiert ist.
- Du rufst Q-BASIC von WINDOWS aus auf und erstellst deine Zeichnung.
- Mit der Taste DRUCK kopierst du den Grafikschirm in die Zwischenablage.
- Du wechselst mit STRG+ESC in die Taskliste und rufst das in WINDOWS integrierte Zeichenprogramm Paintbrush auf.
- Mit *Datei - Einfügen* (oder UMSCHALTTASTE+EINFG) holst du den Inhalt der Zwischenablage in dieses Zeichenprogramm. Du kannst die Zeichnung noch bearbeiten (z.B. einen geeigneten Ausschnitt wählen, die Größe ändern u.a.) und mit *Datei - Drucken* den Ausdruck starten.

Auf diese Weise erhältst du eine getreue Wiedergabe deiner Zeichnung auf Papier. Ein Problem bleibt: Q-BASIC zeichnet mit weißem (buntem) Stift auf schwarzer Hintergrundfarbe und genau so wird die Zeichnung gedruckt (was viel Tinte oder Tuner kostet und nicht schön aussieht). So schaffst du Abhilfe: Füge deinen Zeichenprogrammen den Befehl PAINT am Anfang hinzu:

```
SCREEN 11   'VGA-Schirm, s/w
PAINT (90,110), 1  ' die Koordinaten eines Punktes angeben, die Hintergrundfarbe 1 (weiß) wählen
```

Den eigentlichen Zeichenbefehlen, wie LINE, PSET, CIRCLE usw. noch die Farbangabe 0 (schwarz) hinzufügen, z.B.

```
LINE (100,100) - (400,280), 0
```

Wenn du mit dem Igel arbeitest: Igelprogramm in Kapitel 4 so abändern:
In die Prozedur 'igel' den PAINT-Befehl, gleich nach SCREEN 11, CLS, aufnehmen. In den Prozeduren 'vw' (vorwärts) und 'zeichneigel' jeweils die LINE-Befehle um das Farbattribut 0 (schwarz) ergänzen.

Falls du nicht mit Windows arbeiten kannst, muss auf der DOS-Ebene das Hilfsprogramm GRAHICS geladen sein (evtl. mit dem Parameter /r). Der Druck wird mit der DRUCK-Taste ausgelöst.

Probleme mit Q-BASIC

Vor allem dem Anfänger unterlaufen immer wieder Fehlbedienungen, auf die Q-BASIC oft ungewöhnlich reagiert. Das ist weiter nicht gefährlich, du kannst schlimmstenfalls mit einem RESET (entsprechender 'Knopf' am Computer oder STRG+ALT+ENTF) den Computer neu starten und Q-BASIC wieder aufrufen. Dabei geht allerdings das gerade bearbeitete Programm verloren. Gewöhne dir also an, deine Programme, auch wenn sie noch unfertig sind, zu speichern; du kannst sie dann im beschriebenen Fall laden und weiterbearbeiten..

Problem	**Abhilfe**
1) Der Ausgabebildschirm verschwindet sofort.	Mit F4 wechselst du zwischen Ausgabeschirm und Arbeitsfläche. Füge eine SLEEP-Befehl an geeigneter Stelle deines Programms ein.
2) Nach einer Programmkorrektur bleibt das Programm immer an derselben, inzwischen korrigierten Stelle, hängen.	Starte das Programm neu mit HOCHSTELLTASTE+F5 (nicht nur F5) oder über das Menü: *Ausführen - Start.* Falls das nicht hilft: Programm abspeichern, Q-BASIC beenden und neu aufrufen.
3) Q-BASIC führt ein neues Programm nicht aus, sondern immer noch ein altes, bereits gelöschtes Programm.	Programm abspeichern, Q-BASIC beenden und neu aufrufen.
4) Q-BASIC ist 'abgestürzt' (passiert z.B. manchmal bei *Datei - Beenden*)	Hier hilft nur ein RESET (mit RESET-Knopf am Computer oder: STRG+ ALT+ENTF)

Die wichtigsten Q-BASIC-Schlüsselwörter

Es werden die in diesem Buch angesprochenen Schlüsselwörter kurz erläutert. Dabei wird im allg. nicht der volle Umfang des jeweiligen Befehls dargestellt. In der Hilfe kann der interessierte Benutzer die vollständige Beschreibung des jeweiligen Schlüsselworts finden.

Ein- und Ausgabeanweisungen: PRINT, INPUT, CLS

PRINT bewirkt die Ausgabe der angegebenen Konstanten oder Variablen auf dem Bildschirm oder in eine Datei. Werden hinter PRINT mehrere Druckelemente angegeben, müssen diese durch ein Semikolon '`;`' oder Komma '`,`' getrennt sein. '`;`' bewirkt lückenloses Ausgeben, (wobei für Zahlen die vorderste Stelle immer für das Vorzeichen reserviert ist) '`,`' wirkt wie ein Tabulator. PRINT ohne nachfolgendes Druckelement bewirkt einen Zeilenvorschub ("druckt" also eine Leerzeile).

`PRINT 18`	Die Zahl 18 wird am Anfang der nächsten Zeile gedruckt. (Sofern ein vorheriger PRINT-Befehl nicht mit '`,`' oder '`;`' endete.)
`PRINT x`	Der Inhalt des Speichers x (die Belegung der Variablen x) wird am Anfang der nächsten Zeile ausgedruckt. (Sofern ein vorheriger PRINT-Befehl nicht mit '`,`' oder '`;`' endete.)
`PRINT "x = ";x`	Der Text 'x = ' wird ausgedruckt, unmittelbar darauf folgt der Inhalt des Speichers x.

`PRINT a,b,c` Die Belegungen der Variablen a, b und c wird jeweils am Anfang des nächsten Ausgabebereichs (Spalten der Breite 14) ausgegeben.

`LPRINT` gibt auf dem Drucker aus

`PRINT USING` bewirkt eine formatierte Ausgabe

```
PRINT USING "Artikelnr. #####    Preis: ###.## DM"; artikelnr, preis
```

Die Ausgabe findet genau in der durch den Formatstring festgelegten Form statt, die Artikelnummer darf höchstens 5-stellig sein, der Preis höchstens 3 Stellen vor dem Komma aufweisen.

Ausgabe in Dateien: s. Dateiverwaltung

INPUT liest Eingaben von der Tastatur oder einer Datei. LINE INPUT liest eine Zeile mit bis zu 255 Zeichen von der Tastatur oder einer Datei ein. LINE INPUT ist nur zur Eingabe von Text geeignet.

`INPUT x` Das Programm stoppt, zeigt ein Fragezeichen und erwartet die Eingabe einer Zahl, die der Variablen x zugeordnet wird. Nach Drücken der EINGABE-Taste wird der Programmlauf fortgesetzt. (Achtung: Zahlen mit Dezimalpunkt eingeben, Eingabe eines Kommas bewirkt einen Fehler.)

`INPUT "Dein Alter = ",alter` Statt des Fragezeichens wird der in Anführungszeichen eingeschlossene Text gezeigt.

Bei der Eingabe von Texten ist LINE INPUT vorzuziehen, da alle Textzeichen (auch Kommas) bis zum 'carriagge return'-Zeichen (Drücken der EINGABE-Taste) gelesen werden.

Eingabe aus Dateien: s. Dateiverwaltung

`CLS` löscht den Bildschirm und setzt den Cursor in die linke obere Ecke

Steueranweisungen

Schleifen (Wiederholungen)

FOR - TO - STEP - NEXT

Die Anweisungen innerhalb der FOR-NEXT -Struktur werden so oft ausgeführt, wie die Laufvariable bestimmt.

Beispiel:

```
FOR i=10 TO 100 STEP 2
  PRINT i;
NEXT i
```

druckt die geraden Zahlen von 10 bis 100

WHILE - WEND

Die Anweisungen innerhalb der Schleife werden ausgeführt, solange die ‘Eintrittsbedingung’ erfüllt ist.

Beispiel:

```
INPUT "Noch ein Spiel? (j/n) ";antw$
WHILE antw$<>"j" AND antw$<>"n"
  INPUT "Noch ein Spiel? (j/n) ":antw$
WEND
```

DO - LOOP UNTIL

Die Anweisungen innerhalb der Schleife werden ausgeführt, bis die 'Austrittsbedingung' erfüllt ist.
Beispiel:

```
DO
  INPUT "Noch ein Spiel? (j/n) ":antw$
LOOP UNTIL antw$="j" OR antw$="n"
```

Eine andere Möglichkeit Schleifen zu programmieren, bietet die DO .. LOOP - Anweisung: (s. Hilfe)
Beispiel

```
DO
 INPUT "Noch ein Spiel? (j/n) ":antw$
 IF antw$="j" OR antw$="n" THEN EXIT DO
LOOP
```

Bedingungen

Die normale **zweiseitige Auswahl:** IF - THEN - ELSE - END IF

Beispiel:

```
INPUT "1 oder 2 eingeben ",x
IF x=1 or x=2 THEN
 PRINT "OK"
ELSE
 PRINT "Nicht erlaubt"
END IF
```

Mehrseitige Auswahl: IF - THEN - ELSEIF - ELSE - END IF
Beispiel:

```
INPUT i
IF i<10 THEN
  PRINT "zu klein"
ELSEIF i<15 THEN
 PRINT "gut"
ELSE
 PRINT "zu groß"
END IF
```

Die Bedingungen werden der Reihe nach überprüft, sobald eine zutrifft, wird die folgende(n) Anweisung(en) ausgeführt und die Struktur dann verlassen.

Einseitige Auswahl: IF - THEN - END IF

```
IF bedingung THEN
  anweisungen
END IF
```

Wenn hinter THEN nur *eine* Anweisung folgt, ist die einzeilige Kurzform (ohne END IF) möglich:

```
IF zahl=6 THEN PRINT "Gewonnen"
```

Funktionen und Prozeduren

FUNCTION - END FUNCTION
ermöglicht eigene benutzerdefinierte Funktionen. Auf das Schlüsselwort FUNCTION folgt der Funktionsname und in Klammern die Liste der Parameter, von denen der Funktionswert abhängt. Falls eine Zeichenkette zurückgegeben wird, muss der Funktionsname das Suffix '$' haben. Zwischen den Schlüsselwörtern steht der Anweisungsblock, in dem der Funktionswert bestimmt wird und die Anweisung Funktionsname=Funktionswert. Der Aufruf der Funktion erfolgt im Hauptprogramm oder im Direktmodus durch Nennen ihres Namens. Funktionen und Prozeduren werden in eigene 'Fenster' geschrieben.
Beispiel:

```
FUNCTION fakultaet(n)
  p=1
  FOR i=1 to n
    p=p*i
  NEXT
  fakultaet=p
END FUNCTION
```

berechnet die Fakultät einer natürlichen Zahl.
Aufruf: PRINT fakultaet(6)
Ausgabe: 720

SUB - END SUB
Auf das Schlüsselwort SUB folgt der Name des Unterprogramms (Prozedur) und (evtl.) in Klammern die Liste der Parameter, von denen die Prozedur abhängt. Die Variablen in Prozeduren und Funktionen sind nur innerhalb der jeweiligen Prozedur (Funktion) bekannt (lokale Variable). Umgekehrt 'kennt' eine Prozedur/Funktion nur die Variablen, welche innerhalb des Prozedur/Funktionsblocks definiert oder in der Parameterliste übergeben wurden. Soll eine Prozedur/Funktion auf andere Variable des Hauptprogramms/anderer Unterprogramme Zugriff erhalten, müssen diese mit dem Schlüsselwort 'SHARED' der Prozedur/Funktion bekannt gemacht werden.
Beispiel:

```
SUB warte(t)
 zeit=TIMER
 DO
   x=TIMER
   IF x>zeit+t THEN EXIT DO
 LOOP
END SUB
```

Aufruf im Hauptprogramm z.B. mit warte 3
(Klammern nicht notwendig)
Wirkung: Das Programm wartet 3 Sekunden
(außerhalb der Prozedur sind die Variablen *zeit* und *x* nicht 'bekannt')
(Die Anweisung SLEEP 3 hat dieselbe Wirkung)

Standardfunktionen

`ABS(x)`	Absolutbetrag von x
`INT(x)`	größte ganze Zahl, unterhalb von x
`SQR(x)`	Quadratwurzel von x
`SIN(x), COS(x), TAN(x)`	die trigonometrischen Funktionen (x im Bogenmaß)
`ATN(x)`	Arcus Tangens, Ergebnis ist ein Wert im Bogenmaß, Umkehrfunktion zu TAN
`LOG(x)`	natürlicher Logarithmus von x
`EXP(x)`	Exponentialfunktion mit Basis e, Umkehrfunktion zu LOG
`TIMER`	gibt die Zeit seit Mitternacht zurück (in Sekunden, auf Hundertstel)

Funktionen im Zusammenhang mit Strings (Zeichenketten):

`a$=CHR$(x)`	a$ wird das Zeichen mit dem ASCII-Code x zugewiesen
`x=ASC(a$)`	x wird der ASCII-Wert des ersten Zeichens von a$ zugewiesen
`l=LEN(a$)`	l erhält die Länge der Zeichenkette in der Variablen a$ zugewiesen

MID$ wird als Anweisung bzw. als Funktion verwendet um Teilzeichenketten aus einem String herauszunehmen bzw. um in einem String einen Teilstring zu ändern.

`b$=MID$(a$,i,k)`	b$ ist die Teilzeichenkette der Länge k von a$, beginnend mit dem i-ten Zeichen von a$.
`MID$(a$,i,k)=b$`	In der Zeichenkette a$ wird die Teilzeichenkette der Länge k ab dem i-ten Zeichen durch b$ ersetzt.

Beispiele:

`b$=MID$("Segelboot",6,4)`	b$ erhält den String "boot" zugewiesen
`a$="Segelboot"`	
`MID$(a$,1,5)="Ruder"`	a$ enthält die Zeichenkette "Ruderboot".

INSTR	überprüft, ob ein String Teil eines anderen ist
`x=INSTR(a$, b$)`	x wird die Anfangsposition von b$ als Teilstring von a$ zugewiesen (0, falls b$ kein Teilstring von a$ ist).
`x=INSTR("Segelboot","boot")`	x wird der Wert 6 zugewiesen.

RANDOMIZE und RND	erzeugt (Pseudo-)Zufallszahlen
`RANDOMIZE TIMER`	Der Anfangswert des Zufallszahlengenerators wird bestimmt durch die Zeit, die seit Mitternacht verstrichen ist.
`RANDOMIZE 5`	Der Anfangswert des Zufallszahlengenerators wird auf 5 gesetzt. Dies bewirkt, dass bei Benutzung von RND jeweils dieselbe Zahlenreihe gebildet wird.
`x=RND`	x ist die nächste Zufallszahl (zwischen 0 und 1) der Folge

Arithmetische und Logische Operatoren, Vergleichsoperatoren

+, -, *, /	die normalen arithmetischen Operatoren
`x \ y`	Ganzzahldivision von x durch y, z.B. 20\3 = 6.
`x MOD y`	Rest bei der Ganzzahldividsion von x durch y z.B. 20 MOD 3 = 2 (x und y werden bei \ und MOD zu ganzen Zahlen gerundet)
`NOT`	Verneinung
`AND, OR`	logisches ‘und’ bzw. logisches ‘oder’

Beispiel:

```
INPUT x
IF x>10 AND x<20 THEN PRINT x;"liegt zwischen 10 und 20"
```

`=, <, >, <=` oder `=<, >=` oder `=>` die üblichen Größenvergleiche (in Bedingungen)

Anmerkung: ‘=’ hat zwei Bedeutungen: einmal als Wertezuweisung, einmal als Vergleichsoperator.

Datentypen, Vereinbarung von Variablen, Felder

Q-BASIC kennt die Datentypen INTEGER, LONG, SINGLE, DOUBLE und STRING.

INTEGER (Suffix %):	ganze Zahlen von -32 768 bis 32 767
LONG (Suffix &)	ganze Zahlen von -2 147 482 648 bis 2 147 482 647
SINGLE (Suffix ! bzw kein Suffix)	Fließkommazahlen mit 7 signifikanten Stellen
DOUBLE (Suffix #)	Fließkommazahlen mit 15 signifikanten Stellen
STRING (Suffix $)	Beliebiger Text mit bis zu 32 767 Zeichen

Beispiel:

```
a = SQR(2): b# = SQR(2): c% = SQR(2)
d& = 2^31-1: e$ = "Zeichenkette"
PRINT a, b#, c%, d&, e$
```

Die Variablen brauchen durch kein Suffix gekennzeichnet zu werden, wenn sie zuerst durch eine DIM-Anweisung vereinbart werden.

Beispiel:

```
DIM b AS DOUBLE, c AS INTEGER, d AS LONG, e AS STRING
a =SQR(2) : b = SQR(2) :c = SQR(2)
d = 2^31-1 : d = "Zeichenkette"
PRINT a, b, c, d, e
```

Durch Hinzufügen des Schlüsselworts SHARED werden Variable als **globale** Variable (d.h. jede Prozedur hat Zugriff auf dieses Variable) vereinbart.

Beispiel:

`DIM SHARED a AS INTEGER`	oder kürzer
`DIM SHARED a%`	(Diese Anweisung muss im Hauptprogramm, möglichst am Anfang, stehen.)

Felder

`DIM`	reserviert im Speicher Platz für Felder oder vereinbart eine Variable
`DIM a(10), b(20)`	ein (numerisches) Feld **a** mit den Indizes von 0 bis 10 und ein Feld **b** mit den Indizes von 0 bis 20 wird geschaffen
`DIM zahlen(5 TO 12)`	Die Indizes für das Feld **zahlen** laufen von 5 bis 12.
`DIM matrix(10,15)`	Ein Zahlenfeld mit 11x16 Elementen wird geschaffen
`DIM namen$(10)`	Ein Feld von 11 Stringvariablen name$(0) bis name$(10) wird geschaffen.

Dateiverwaltung

(Wir beschränken uns auf sequentielle Dateien)

OPEN, CLOSE Öffnet (schließt) eine Datei auf einem externen Gerät, im allg. einem Laufwerk. Es können gleichzeitig mehrere Dateien geöffnet sein, die an ihrer Dateinummer unterschieden werden.

OPEN Dateiname FOR OUTPUT-INPUT-APPEND AS #Nr

Beispiele:

`OPEN "adressen.dat" FOR OUTPUT AS #1`	
`PRINT #1, name$`	Die sequentielle Datei "adressen.dat" wird auf dem aktuellen
`PRINT #1, ort$`	Laufwerk /Verzeichnis geöffnet (z.B. neu angelegt) um Ausgaben

`CLOSE #1` des Programms aufzunehmen, die Dateinummer ist 1. Während des weiteren Programmlaufs bleibt die Dateinummer 1 für diese Datei reserviert, bis sie wieder mit dem Befehl CLOSE #1 geschlossen wird. In eine mit OUTPUT oder APPEND geöffnete Datei wird mit PRINT #Nr geschrieben. (APPEND statt OUTPUT wird verwendet, wenn eine auf der Diskette/Festplatte bereits existierende Datei erneut geöffnet wird, um weitere Daten am Ende anzufügen.)

```
OPEN "texte.dat" FOR INPUT AS #2
WHILE NOT EOF(2)
  LINE INPUT #2, text$
  PRINT text$
WEND
CLOSE #2
```

Die Datei "texte.dat" im aktuellen Verzeichnis wird zur Eingabe ins Programm unter der Dateinummer 2 geöffnet. Die einzelnen Datensätze der Datei sind durch ein 'carriage return' - CHR$(13) - getrennt und werden nacheinander eingelesen und auf dem Bild- schirm ausgegeben, bis das Ende der Datei (EOF) erreicht ist

Graphikbefehle

`SCREEN`	legt den Grafikmodus fest, z.B.
`SCREEN 11`	VGA-Schirm mit 480x640 pixels
`SCREEN 9`	EGA-Schirm mit 350x640 pixels
`WINDOW(x1,y1)-(x2,y2)`	legt ein Koordinatensystem mit unterer linker Ecke (x1,y1) und oberer rechter Ecke (x2,y2) zugrunde. Damit maßstabsgerecht gezeichnet wird, sollte das Verhältnis x2-x1 zu y2-y1=4:3 sein (beim EGA-Schirm 64:35).
`LINE(x1,y1)-(x2,y2)`	zeichnet eine Strecke von (x1,y1) zu (x2,y2)
`PSET(x,y)`	setzt einen Punkt an der Stelle (x,y)
`CIRCLE(x,y),r`	zeichnet Kreis vom Radius r um (x,y)

Weitere Befehle

`END`	Ende des Programms (kann meist weggelassen werden).
`SLEEP 5`	Der Programmlauf wird für 5 Sekunden unterbrochen. Wird der Parameter weggelassen, hält das Programm, bis eine Taste gedrückt wird.
`REM`	Kommentarzeichen am Beginn einer Zeile. Diese Zeile wird im Programmlauf ignoriert (kann durch ' ersetzt werden)
`'`	der auf ' folgende Rest der Zeile wird ignoriert.
`LOCATE n,m`	Der Cursor wird in Zeile n und Spalte m positioniert, eine anschließende Ausgabe mit PRINT findet an dieser Stelle statt.
`SWAP a,b`	Vertauscht den Inhalt der Variablen a und b.
`INKEY$`	fragt die Tastatur ab und gibt das gerade gedrückte Zeichen zurück. (Gibt eine Nullfolge zurück, wenn keine Taste gedrückt ist.)

Beispiel

```
PRINT "Leertaste drücken um anzuhalten"
DO
LOOP UNTIL INKEY$=CHR$(32)' 32 ist der ASCII-Code der Leertaste
```

LITERATUR

1) Barth P., Nicolai K.-H.: Kreativ mit Q-BASIC
Bonn, Dümmler 1995

2) Baumann R.: Computerspiele und Knobeleien programmiert in BASIC
Würzburg: Vogel 1984

3) Engel A.: Elementarmathematik vom algorithmischen Standpunkt
Stuttgart: Klett 1977

4) Kebschull, G.: Richtig einsteigen in QBasic
München, Microsoft Press Deutschland, 1992

5) Monadjemi, P.: Visual Basic 3.0 für Windows
München: Markt und Technik, 1993

Stichwortverzeichnis

Abbruchbedingung 22
ABS 134
Achsenkreuz 51
AND 38, 132
Array 86
ASC 76, 135
ASCII-Code 75/78
ATN 134

Bildschirmlöschen s. CLS
Bildschirmmodus 47
Bit 75
Byte 75

Cäsarchiffre 74 ff
Chiffrieren 74 ff
CHR$ 76, 135
CIRCLE 50, 137
CLOSE 80, 136
CLS 10, 12, 122
COS 134

Datei 79/80, 94 ff, 136
Dateiname 79/80
Dateinummer 80, 94, 136
Datentyp 136
DIM 86, 136
DIM .. AS .. 37, 136
DIM SHARED 35, 136
Direktmodus 9 ff
DO ... LOOP 24, 133
DOUBLE 136

EINFG-Taste 7, 130
EINGABE-Taste 7, 10
ELSE 25
ELSEIF 27, 130
END 12
END IF 25
ENTF-Taste 7, 130
Entscheidung 25 ff
EOF 81
Eratosthenes 88
EXP 134

Farben (Grafikschirm) 48
Feld 86, 136
File 94
Flagge 39
FOR ... NEXT 18, 23, 132
FUNCTION ... END FUNCTION 32, 134
Funktionen 31 ff
- selbstdefinierte 31 ff
- rekursive 96 ff
- Standard 134
Funktionsgraph 53
Funktionstasten 8, 130

Ganzzahldivision 37, 135
Grafikbildschirm 47

Hilfe 9

IF .. THEN 26, 133
IF .. THEN .. ELSE .. END IF 25, 133
igel 59, 60
INKEY$ 137
INPUT 13, 16, 132
INSTR 82, 135
INT 31, 134
INTEGER 37, 136

Koordinatensystem 50
Kreis 50, 63 ff

Laden von Datendateien 81, 94, 136
Laufbedingung 22
LEN 70, 135
li (links) 59, 60
LINE 47, 137
LINE INPUT 79
LOCATE 51, 137
LOG 134
LONG 136
LOOP 24, 133
LPRINT 69/70

Magic Eye 56, 120
Menü 36
MID$ 71, 135
MOD 37, 135

NEXT (s. FOR)
NOT 135

Öffnen einer Datendatei
- zum Schreiben 80, 94, 136
- zum Lesen, 81, 94, 136
OPEN 80, 136
OR 135
Ordnen 92 ff
OUTPUT 80

Palindrom 72
Pixel 48
Potenzieren 10
PRINT 10, 16, 131
PRINT USING 20, 132
Prozeduren 34, 62
PSET 48, 137

RANDOMIZE 29, 135
re (rechts) 59, 60
Rechteck 49
Rekursion 96 ff
REM 12, 137
RND 29, 135
RW (rückwärts) 59, 60

Schleifenrumpf 18
SCREEN 47, 137
SHARED 35, 70, 136
Simulation 29, 87
SIN 134
SINGLE 136
SLEEP 137
Speicher 12
Speichern von Daten 80, 94, 136
Spezifikation 64
SQR 134
STEP 23, 132
STRING 136
String 70, 135
Struktogramm 17, 18, 22, 23,25
su, sh 59, 60
SUB ... END SUB 34, 134
Suchen 81/82
SWAP 137

Teiler 37
- größter gemeinsamer 38, 40
Teilstring 71
Text 11
Textvariable 68
TIMER 29, 135
Trennzeichen 11
turtle 59

Unterprogramm 34 ff, 62
UNTIL-Schleife 22

Variable 13
- lokal 35
- global 35
vw (Vorwärts) 59

Wertezuweisung 13, 17
WHILE ... WEND 21, 132
WHILE-Schleife 21
Wiederholung 18 ff, 21, 22, 24
WINDOW 48, 137

Zählvariable 18
Zeichenkette 70
Zufallszahlen 29 ff, 73, 135

COMPUTER-PRAXIS MATHEMATIK

- **Dieses Lehr- und Übungsbuch**
  - ist entstanden aus zahlreichen Lehrerfortbildungsveranstaltungen und Praktika mit Lehramtsstudierenden;
  - wendet sich an Schülerinnen und Schüler, Studierende und Lehrende der Sekundarstufe I und ist sowohl als Unterrichtswerk als auch zum Selbststudium geeignet;
  - enthält Programme aus den Bereichen;
    Mathematik,
    Koordinatengrafik,
    Igelgrafik,
    Spiele,
    Textanalyse,
    Dateienverwaltung.

- **Dieses Arbeitsbuch**
  - bereitet die jeweiligen Inhalte anhand zahlreicher, nach Umfang und Schwierigkeit abgestufter Arbeitsaufträge auf;
  - vermittelt den Lernenden die Grundzüge einer modernen, anweisungsorientierten Programmiersprache und die Denkweisen des strukturierten Programmierens;
  - bringt zu den meisten der 327 Aufgaben die Lösung im Anhang.

- **Dieses ansprechende Mitmach-Buch**
  - fordert ein aktives Lernen an vielfältigen Problemen;
  - baut aus einfachen Aufgaben aus verschiedenen Gebieten durch Variation und Erweiterung der Aufgabenstellung immer komplexere Algorithmen auf;
  - macht die Grundstrukturen, aus denen sich Algorithmen zusammensetzten, durchsichtig.

- **Diskette**
  Zu diesem Buch gibt es eine Diskette, die sämtliche Lösungen der 327 Aufgaben und zusätzliche Programmbeispiele enthält, vgl. Seite 2.

- **Die zweite Auflage** berücksichtigt die neue Rechtschreibung.

ÜMMLERbuch 4532